Makia Minich (makia@llnl.gov)

The GeDI Document

May 7, 2012

Contents

1 Information

GeDI (the GEneric Diskless Installer) is a tool to help aid in the setup of diskless Linux clients. Made up of a set of shell scripts (most of which written in bash) the goal is to provide an easy way to configure and manage one or greater Linux systems with minimal administrative overhead. This document is expecting at least some knowledge of the idea of network booting a system as well as a base knowledge of Linux systems administration. Because each system can handle network booting differently, Google[1] is a good place to look if you need more information on how this works.

We'd recommend reading through this entire document before you start working on your image. Primarily because we took the time to write it all, but also because it might answer questions that you didn't even know you had.

2 The Hardware Setup

GeDI is specifically written for a management/client model. Namely, there is a specified management server whose sole duty is to run the clients, and a set of clients who allow for interaction from users. Depending on the number of clients you are looking to implement, your management node should be able to handle them accordingly. Therefore, the biggest components to focus on are network speed, memory size and processor speeds. In addition, a large enough hard drive to hold the management node's root filesystem as well as the clients' root filesystem is a must if you plan on using an NFS export from the management node. In an ideal setup, your mangement system should have some redundancy options available, such as RAID-1 (for mirroring). Not much is required from the clients perspective. They must have the ability to network boot in some fashion (e.g. PXE-boot), and it is usually a good idea to have a fair amount of memory available for user interaction, keeping in mind that the root filesystem may need to use some of this memory.

Table 1 shows the setup that we're using (which will be the example throughout this document). Not much is really needed from the hardware standpoint. The most important component for the entire setup is a stable and reliable network. Without this the clients will have a hard time talking to the management node, and could render them completely useless.

Table 1: System Layout

Management Node	4G Ram Dual Processor Xeon 2.8Ghz (hyperthreading enabled) Gigabit Ethernet 4 40G SCSI Hard Disk Drives (Hardware RAID-1 in two pairs)
Clients (1 or more)	1G Ram Single Processor Pentium 4 2.8GHz 10/100baseT Ethernet no hard disk drive
Network Equipment	10/100baseT Switch with GigE uplink
Serial Console	Used to manage the management node

3 The Software Setup

GeDI consists of one server piece, the GeDI rpm, and a client piece. The server peice contains all of the tools needed to create an image for a diskless client and configure it for use. The client piece (which

[1] http://www.google.com

should be installed in the image) contains scripts to modify the image into something useable after boot. In addition, you will need a few other linux tools to succesfully network boot and load your client nodes. Included are:

- dhcp-2.0pl5-8 or any better dhcp server
- tftp-server-0.28-1 or any better tftp server
- xinetd for the tftp-server
- nfs-utils for nfs exports
- rpm-4.0.2-8 or anything higher

Most of these are relatively standard issue for an RPM based linux distribution, and should be readily available. In addition, your management node should have a repository of all of the RPMS that you wish to install onto your diskless image. GeDI, by default, will look in `/usr/gedi/RPMS` for these RPMS (in fact, the first RPM one should put in this directory is gedi-client).

4 The Directory Structure of GeDI

When the GeDI RPM is installed, a number of directories and files get created on the management node. Please see table 2 for the list.

Table 2: GeDI Directory Structure on the Management Node

`/etc/gedi`	Directory containing the MAC.info file, rpmlists and variables files.
`/usr/gedi/local`	Directory where the client's local customizations go.
`/usr/gedi/nfsroot`	Directory where the client's image will be created.
`/usr/gedi/patches`	Patches that should be applied to the installing image.
`/usr/gedi/pxelinux.0`	Boot file sent to the client aiding in PXE-booting.
`/usr/gedi/RPMS`	A repository of RPMS that are available to be installed on the client system.
`/usr/gedi/scripts`	Scripts used to create the images.
`/usr/gedi/tools`	Scripts made available as tools to help manage the system.

For you, the most important of these directories are etc, RPMS, scripts and tools. These are the places where you'll do most of your work in configuring the system. One important thing to note here is that the gedi RPM is actually relocatable. So, if you don't happen to like `/usr/gedi` (or you want to use a bigger disk mounted in some other location) then you can pass the "`--prefix`" option to RPM on the install line. For example:

```
# rpm -ivh --prefix /tftpboot gedi-0.4.0-1.noarch.rpm
```

If you don't want to relocate the RPM install, another option is available as well. Each location of the directories is controlled by a variable in the `/etc/gedi/variables` file. So if you, perhaps, wanted to place your RPMS in /usr/local/RPMS, you can change the associated variable in the `variables` file.

On the client side, you'll want to install the `gedi-client` RPM. See table 3 for the list of directories.

5 A Note About Client Types

GeDI has been created with the idea that you could be serving different types of clients from a single management server. For example, if you have some clients where people would be compiling a lot of

Table 3: GeDI Directory Structure in the Client Image

`/etc/gedi`	Directory containing the `gedi_client.conf` configuartion file.
`/sbin/mkinitrd_gedi`	Script to create the ramdisk for GeDI.
`/usr/share/gedi/scripts`	Support scripts to get the image running.
`/usr/share/gedi/src`	UnionFS source files.

code, and other clients where this is restricted you could easily create two node types–e.g. `compile`, `not_compile`. Every script that works on an image will require you to supply this node type on the command line, so you might want to think about what you'd like to use before jumping into things. In the next section, we'll actually see how this node type is used.

6 Creating Your First Image

6.1 The RPM List

GeDI is based on an NFSroot style setup of a diskless client. In short this means that, for each of the clients, the root filesystem will be made available over NFS instead of found locally on the hard drive. For this to work, we'll need an NFS exported directory from the management node that contains all of the elements of a working system. This can be accomplished through the use of an RPM list, and the "`create_nfsroot`" script. The RPM list is made up of all of the RPMS that you wish to install on your system, and looks something like:

```
Canna-libs-3.5b2-62.i386.rpm
ElectricFence-2.2.2-8.i386.rpm
Glide3-20010520-13
LPRng-3.8.9-4*
```

Each line is made up of one RPM, and is comprised of it's filename (GeDI can handle file globbing and name expansion). The actual sorting of the list doesn't really matter (though it has been found that properly sorted lists tend to give less errors when they are being installed). If you have a running system that you'd like to base your client off of (which is the easiest thing to do) you can run the following command on that system:

```
# rpm -qa --queryformat "%{NAME}-%{VERSION}-%{RELEASE}.%{ARCH}.rpm\n" > /tmp/rpmlist
```

This will put (into the file `/tmp/rpmlist`) the list of all of the RPMS that you have on your running system. This file can then be directly used by GeDI to create your image.

Once you have the list all setup, you can put it into the `/etc/gedi` directory. If you have determined a node type for this RPM list, you will want to name the file `rpmlist.<nodetype>` (e.g. `rpmlist.other`). The "`rpmlist`" portion of the name is unimportant, but it's a bit easier to call it "`rpmlist`" to know what it is. On the otherhand, the "`.<nodetype>`" portion is quite important as it will tell the `create_nfsroot` script that you are looking to create an image for a specific nodetype (`<nodetype>`), so be careful of what you call it.

> ****NOTE****
>
> **This is a pretty good time to mention a little bit about the client's kernel. The latest version of GeDI makes use of an initial ramdisk to allow you to boot any**

available kernel on your clients. Put slightly different, we no longer require a special re-built kernel for the client to boot up. When you install a new kernel into the image, you'll want to run `/usr/gedi/scripts/create_ramdisk` (which is automatically run from the `configure` script). This will go inside of the image, and create a ramdisk for each kernel you have available. This ramdisk will contain any useful network drivers and have the ability to NFS mount your root filesystem. In addition, the script will create the `gedi_pxe` and `boot.msg` files in the images `/boot` directory which are used to allow you a way to select which kernel to boot on while the system is coming up (much like grub or lilo on diskfull systems). ********

Before you can count your RPM list as finished, make sure that you add the `gedi-client` RPM. This contains useful scripts that will get your system booted and running (including the ability to create ramdisks).

6.2 Test Your RPM List

Once you have your RPM list all set up, it's time to test your list. This step is not required, but highly recommended since it might save some time in the long run. To test your RPM list, you can run the following command:

```
# /usr/gedi/scripts/create_nfsroot -t <RPM List>
```

If you happened to put your rpmlist in `/etc/gedi`, you can substitue `<RPM List>` with just the name of the file, otherwise you'll have to give the full path to the file. Remember, though, that the filename must end with "`.<nodetype>`" or things might get a bit confusing. After a little bit of time, you should see what appears to be your RPM's installing. Actually, what is happening is this:

1. We check your RPM list to make sure that all of the files exist.
2. We perform a chroot'ed RPM-database install of the selected RPM's.
3. We remove our test directory.

This database install is a simple way to check if all of the dependencies for your RPM list are met. No actual RPM's are installed but the test RPM database is told that they are.

6.3 Customize Your Image

To aid in the local customization of your image, we have created the `/usr/gedi/local` directory. This directory should contain copies of all of the files that you would like to have installed in your image as well as a configuration file (`local.conf`) to detail their permissions and destination. The `local.conf` file has the following layout:

```
<local filename>    <image>   <destination>      <mode>    <user>    <group>
```

The `<local filename>` is the name of the file in `/usr/gedi/local`. `<image>` is the name of the image that the file should be installed into (using "*" will install the file into all of the images). `<destination>` is the full path and resulting filename where it should be copied to. `<mode>` is the mode that will be passed to chmod for setting the correct permissions on the file. `<user>` and `<group>` are used to set ownership on the file. For example, to copy `hosts.allow` to the compute image as `/etc/hosts.allow` with owner and group of root and permissions of 644, we'd use the following line in `local.conf`:

```
hosts.allow         compute   /etc/hosts.allow   644       root      root
```

Or, to install the same file into all images, we'd use:

```
hosts.allow         *         /etc/hosts.allow   644       root      root
```

This format allows us to keep a flat directory, but keep the final destination for the file known. If, perhaps, you instead wanted to use the management nodes actual `hosts.allow` file (which should be located in `/etc`), you could instead do the following:

```
/etc/hosts.allow    *         /etc/hosts.allow   644       root      root
```

This tells GeDI to actually pull the real file from a different location than the `local` directory.

6.4 Create Your Image

Now, if we see that all of our dependencies are fulfilled and our customizations are in place we are ready to actually create our image. To do this, we can run the following (notice the lack of a '-t'):

```
# /usr/gedi/scripts/create_nfsroot <RPM List>
```

Now, you can probably step away for a bit since this usually takes a little while (though we could possibly go on to the next steps and let this finish in the background).

7 Getting the Server Ready

If all went well, we should now have an image to serve out, but we still need to get everything ready on the managment node. The first, and most important, piece is the MAC.info file. This is a file that lives in `/etc/gedi` and contains a mapping between a hostname, IP-Address, hardware MAC-Address, and node type. The file looks something like this:

```
00:00:00:00:00:00            hostname      192.168.1.1      other
```

`/usr/gedi/tools/MAC_Collector` is a simple script that helps aid in the process of discovering clients. Basically, it listens for broadcasts on your ethernet network, and upon receiving one it asks questions about that node (including whether or not the system found is the client you are expecting). It then performs all of the necessary system configuration. It's quite handy when it works, but the script can be somewhat unreliable on networks that experience heavy traffic. Therefore, you might need to find a more reliable way to collect MAC addresses. If you would like to use a network-based method, you can use the following command (which is the heart of the `MAC_Collector` script):

```
# /usr/sbin/tcpdump -i <Ethernet Device> -c 1 -entl port bootpc 2>/dev/null |
   /bin/awk '{
      split($1, sep, ":")
      printf("%02s:%02s:%02s:%02s:%02s:%02s\n", sep[1], sep[2], sep[3],
                                                sep[4], sep[5], sep[6])
   }'
```

This will return one MAC address and display it on STDOUT. Another method is to actually watch the screen of the network booting system. Generally, when the PXE Boot message appears, along with it comes the MAC Address. Depending on the system, you might need to quickly write it down, as it might not stay on screen for too long.

Another option is to just use ranges in the `MAC.info` file. This layout looks something like:

```
192.168.1.1-192.168.1.10      other
```

The first column contains the IP-Address range that you'd like to use for the node type you defined in the second column. This method has the benefit of being able to boot clients without having to collect MAC addresses, but keep in mind that it will then attempt to listen to any dhcp traffic on your network and try to boot those requestors as well.

After you've got your `MAC.info` file setup the way that you like it, you can run the `config_services` script in `/usr/gedi/tools`, which will perform the remaining actions needed to get your system all set up and ready. The `config_services` script is given as an aid in configuring all of the services that are needed to get GeDI up and going. By default, this script will do the following:

1. Create `/etc/dhcpd.conf`, and populate it with the systems listed in MAC.info.
2. Add entries to `/etc/exports` to NFS export the correct image to each system.
3. Add entries to `/etc/hosts` for each system listed in MAC.info.
4. Restart dhcpd.
5. Re-export all of the NFS exports.

Once all of this is done, your server should be ready to boot your clients.

> ****NOTE****
>
> **The modification of the exports, hosts and other files can be toggled by modifying the variables "`MODIFY_EXPORTS`", "`MODIFY_HOSTS_FILE`", and any other "`MODIFY_`" entries in the file `/usr/gedi/etc/variables`. By default these are turned on, so if you want to control what happens to your exports and hosts make sure and set these to "NO".**
>
> ************

8 Booting Your First Node

At this time, you should be just about ready to go. As a checklist before booting your node, make sure that you have the following services started and configured on the management node:

- NFSD
- dhcpd
- tftpd (usually set through xinetd)

Without these services, you're clients will be dead in the water. So, if everything is running, you should be able to boot up a client and see the following happening:

1. The client powers up and starts going through it's BIOS messages.
2. The client should attempt to start network/PXE booting.
3. The server should receive a broadcast from the client and send some network configuration information to the client.
4. The client will then ask for a file to boot off of.
5. The server will send the pxelinux.0 file to the client as well as the name of the configuration file it should use to boot.
6. The client will load and run pxelinux.0, which basically tells the client to send a request for the appropriate configuration file.
7. The server will send out the related file.
8. The client will read the file and figure out the name of the kernel to ask for.
9. The server will send the correct kernel to the client.
10. The client will boot the received kernel with a specified kernel-command line and linux will start loading.
11. When the root filesystem is about to be mounted, the client will issue an NFS request to the server for the appropriate mount.
12. The client will mount the NFS-root filesystem read-only as well as a writeable RAM disk.
13. The NFS mount and RAM disk are layered into a unionfs filesystem which is used as the root filesystem.
14. The system pivot-roots to the new unionfs filesystem and starts booting the correct runlevel.

If all went well, you should now have a working client to use and enjoy.

9 Managing The Images After Everything Else Is Finished

So, if you've done all of the steps correctly, you should have a set of working diskless clients up and running. Pretty cool, huh? Now comes the time of how to manage these systems.

9.1 Adding Users

First, we'll deal with adding users to the clients. If you happen to standardly use local passwords on systems (or some various form of pam-authentication type) you can use the sync_passwd script in `/usr/gedi/tools`. This simple script will take the passwd (and associated) files from the management node and install them into the images that you are hosting. This can be run from cron on the management node so that all you need to worry about doing is updating one system's passwd entries. If you are worried about restricting users from accessing your management system, you might take a look at the file `/etc/security/access.conf` which is installed by pam. With some tweaking, you can restrict access to your management node, while allowing access to your clients all with the same passwd files.

9.2 Changing Configuration Files

Now, let's say we need to change some configuration files that are being placed into the images. This is easily done by using the `local` directory and the `configure` script. Just place those pesky configuration files into `/usr/gedi/local`, modify the `local.conf` file and re-run `/usr/gedi/scripts/configure` to put them in place. If the client is running, and this is a change that needs to somehow be put into affect (e.g. restarting a network service) you might need to actually perform that action on the clients.

Because GeDI is using unionfs, there's a chance that you'll find that your clients may not see the change that you made on the management node. Generally this will happen if, at some point after booting, the specific file has been modified locally on your client. If this does happen, and you want to fix it, you will need to reboot your system (or just install the new version locally as well).

9.3 Local Modifications - Image

OK, that covers the handling of configuration files, but sometimes we also need some actual filesystem manipulations to happen (e.g. make links to directories, turning off and on various services). This can be done with the special `local_mods` script. `/usr/gedi/scripts/local_mods` (which is run everytime `configure` is run) is a simple (or more complex if you'd like to make it that way) script that will be run before copying files from the local directory. GeDI provides an example `local_mods` script in the `/usr/share/doc/gedi-<version>/examples` directory so that you can take a look at what kind of things can be done. Remember that this script is written from the point of view outside of the image, so you need to be especially careful in what you are doing. The example references the `/etc/gedi/variables` file, which can be helpful in not forcing you to remember the directory structure.

9.4 Local Modifications - Booting System

While the `local_mods` script modifies the system from the image point-of-view, we should also talk about how to modify the image of a booting system. This can be done one of two ways:

- Create a `rc.local` script and place it in the `/usr/gedi/local` directory.
- Modify `/usr/gedi/local/rc.readonly` with the changes that you want.

The `rc.local` is probably the most generic way to handle cusomizations to your system (such as modprobing specific modules and whatnot). This way does work, but remember that `rc.local` is generally the very last script to be run at boot time. The `rc.readonly` script is a RedHat'ism (introduced in RHEL4) that is actually run from `rc.sysinit` early in the boot process. So, if you had a filesystem that you wanted to mount long before the services start running, this is your best bet.

9.5 Image Upgrades

So, now what happens when we want to upgrade the image or even install new packages? Well, there are a few ways to handle this, it all depends on personal preference.

1. **Just do it by hand and damn the consequences.**

   This way makes life in the short-term nice and easy. If you have a new RPM to install, just install it (or upgrade it) and be done with it. This is easy enough to accomplish with the following command:

   To install:

   ```
   # rpm -ivh -r <PATH TO IMAGE> <RPM TO INSTALL>
   ```

   To upgrade:

   ```
   # rpm -Uvh -r <PATH TO IMAGE> <RPM TO INSTALL>
   ```

   To freshen:

```
# rpm -Fvh -r <PATH TO IMAGE> <RPM TO INSTALL>
```

Not too bad, everything should go as planned and the image should have the new RPM. The only problem is that you don't have this change documented in the RPM list. This brings us to the next idea.

2. **Edit the associated RPM list and manually upgrade the image.**

   Basically, we'll do everything from step 1, but make sure that we also make the needed changes in the rpmlist that we use (just in case we need to create a new image for whatever reason). This can be quite handy in the long run and not too difficult to follow through on.

3. **Use some handy scripts to keep things in sync.**

   GeDI comes with a few tools to aid in making sure that your image matches what you think should be installed (at least RPM-wise). The first is `update_rpmdir`. If you have a central repository set up where you are keeping RPM updates, this script can help you keep `/usr/gedi/RPMS` up-to-date as well. As an example, you can run the following:

   ```
   # /usr/gedi/tools/update_rpmdir -u <PATH TO UPDATE RPMS>
   ```

   This will take all of the RPMs found in your updates directory, and copy any newer packages into `/usr/gedi/RPMS` so that they are ready to be put into the image. Once this is done, we can now use the `update_rpmlist` script to get our rpmlist to match what's newly available. For example, you can run the following:

   ```
   # /usr/gedi/tools/update_rpmlist <RPM List>
   ```

   This will compare what packages are available in the RPM directory to the items in your list. If there's anything newer available in the directory, your list will be updated to this new version (any errors should be displayed on STDOUT and probably need to be taken care of before moving on). Now that we have an updated list, we can see what changes need to be made to the actual image. This can be done with the `rpmdiff` script. This script compares what's in the list to what is currently in the image's RPM database and reports any discreprencies:

   ```
   # /usr/gedi/tools/rpmdiff <RPM List> <Nodetype>
   ```

   The script also has the option of modifying your `rpmlist` and `image` with the actual changes that need to happen. This can be done with the `-g` option:

   ```
   # /usr/gedi/tools/rpmdiff -g <RPM List> <Nodetype>
   ```

   Pretty simple, eh? Well, this way is a bit more tedious, but ensures that your RPM repository and your list and your image are all in sync, which long-term can help rule out certain issues if something goes wrong.

9.6 Adding and Removing Clients

Since it is highly unlikely that from the first time you install your system you'll never change what clients are connected, we should probably talk a little bit about how to add and remove clients from GeDI. This isn't actually all that tedious of a task (especially with later versions of GeDI). First, we have to remember that the following files have an affect on what clients get booted and what they boot:

- `/usr/gedi/etc/MAC.info`
- `/etc/dhcpd.conf`

- /etc/exports
- /etc/hosts
- <image>/boot/pxe/gedi_pxe

By default, GeDI's config_services script can handle the modifications of each of the system files based on changes that you make to the MAC.info file. So, as an example, if we wanted to change the MAC address of one client, remove another client, add other client, change an IP address, and also change the options for an existing client, this can all be done by modifying the MAC.info file, and re-running config_services. This is much better than having to go and fix all of the files by hand.

> ****NOTE****
>
> **Remember, the modification of most configuration of the files is dependent on the "MODIFY_" variables set in the /etc/gedi/variables file. If these are set to no, you'll need to modify those files by hand.**
>
> ************

10 GeDI Architecture

Not in the realm of how to make it work, this is more of a what's under the covers section. GeDI was created from the "single image shared among many systems" point-of-view. Therefore, GeDI needed a way to allow one image to be created, but allow the client nodes the ability to modify the image to suit their needs (e.g. sendmail really likes to modify files in /etc/mail). Early incarnations of GeDI use the read-only NFSRoot method utilizing a local ramdisk. Each node would boot up on the read-only root filesystem, and early in the boot process they would create a ramdisk, and start populating it with the files that needed to be written to. While this does work, there is a certain amount of management overhead as you try to keep an eye on every server that needs to modify a file and make sure that the file is available to be written. In addition, this function required that at image creation, certain files were removed and turned into symlinks that pointed into the non-existant ramdisk. This could potentially upset the RPM verification process (as files are modified and removed).

A cleaner solution was found in using UnionFS[2]. This allowed us to take the symlinks to ramdisks completely out of the equation. Instead, what we do is mount the root filesystem read-only (just as before) and create/mount a ramdisk. Utilizing UnionFS, we then merge the two filesystems, layering the writeable ramdisk on top of the read-only NFS mount. The result is then used as the root filesystem, and the system continues booting up. Any files that are modified are automagically copied from the NFS filesystem into the ramdisk and become writeable, allowing for local copies to be used.

Now, there can be an issue with UnionFS that still needs to be addressed (but the same issue also existed with the prior symlink-ramdisk idea). What happens if you want to modify a file in the image, but a client has a local copy that it has written to? There are a couple of ways to deal with this (at the moment). The first is to go ahead and reboot the node(s), on reboot the modifications are lost and you're back to your original image. The second is to remove the local copy from the /union/writeable directory structure, but according to the UnionFS developers this is not a recommended practice.

With later releases of GeDI, you now have a choice as to which method (UnionFS or RAM-based) to use on the client. By default, GeDI will attempt to build and install the UnionFS modules, but if they aren't available, GeDI will instead boot using the RAM-based method (creating writeable copies of /etc and /var).

[2]See http://www.unionfs.org for more information.

11 Conclusion

Surely there's more that needs to be written here, but at the moment this is all that there is.

12 Side Notes

As people start using GeDI (which I hope they do), we'll start adding information to this section on various findings that come up.

12.1 SELinux

SELinux support in GeDI is non-existant and you might see problems with your system caused by SELinux. For example, if you start seeing weird errors while running create_image, there's a good chance that SELinux is trying to "help" you out by not allowing you to write certain files. The way to fix this is to disable SELinux via the kernel command line (e.g. in `grub.conf` add `selinux=0` to the kernel line for your kernel). Also, if you happen to see weird "cannot write" messages from your client (most noticeable is with `ssh`) you should also disable SELinux there. To do this, add `selinux=0` to the `BOOT_OPTIONS` line in the GeDI `variables` file (don't forget to run `configure` when you're done).