

Introduction:

This document will present two different solutions to the pressing problem of computer backups. While the need for backups is not generally debatable, it is not an operation that is particularly engaging, even to the technically inclined. I will assume, therefore, that you have been delegated (perhaps by yourself) to provide this under-appreciated but undeniably vital service, and would like to quickly and painlessly get the job behind you.

Let me begin by disabusing you of the notion that this can be done quickly and painlessly. The avoidance of unpleasant activity will come when one of your users require some backed up information, and you can quickly and painlessly provide it. The installation and configuration of a backup system, however, is a bit tedious, and will probably require more thought and time than you realize up front. Think of it as delayed gratification when you are able to help others recover from potentially catastrophic problems.

In addition, I should mention that this discussion will assume you are using a Linux system as the backup server. To keep expenses down on what is generally viewed as an overhead service, I use open source software in every instance possible. Most modern Linux and BSD UNIX distributions come with a packaging system that probably already offers the backup software I will be discussing, but you might decide to compile the source in order to get some feature in the latest version. In the two example installations I will discuss, one was installed on Scientific Linux, the other on Ubuntu, and while the former is based on RedHat and the latter on Debian, there were no distribution issues. In fact, the installation in both cases borders on the trivial.

Before plunging into specifics, a word about security. In point of fact, the primary mechanism which allows you to restore lost data is redundancy (multiple copies of the same data in different places). Unfortunately, the more copies of data that exist, the more difficult it becomes to secure that data (in the sense of controlling access to the data). In other words, increasing security against loss of data decreases to some extent your ability to keep the data private. In addition, please realize that if someone hacks into your backup server, they don't need access to the systems that it backs up; all the data is conveniently there in one place for them. Many of the "security breaches" you read about on the net involve access to backup media rather than to user computer systems. This argues that you should have a dedicated backup server with few accounts, limited access, and providing no other services.

Backup software:

There are different ways to backup your work done on computer systems, and the optimal solution for you will depend on many things, including budget, number and types of systems to be backed up, rate of the accumulation of data or changes to data and personnel. My experience is that taking the time and trouble to setup a largely automated system (in the sense that backups happen on a schedule without intervention by an administrator) repays the effort by the consistency of backup coverage when something must be restored (ie nobody has to remember when their system was last backed up). In accordance, I will discuss two backup

systems, either of which will serve well within a given set of constraints. Both are open source software systems which may be installed and used without cost or license fees. The two differ in the kinds of environments they target. The first, named bacula, is an "enterprise" system that is used in many installations where there are several networked systems of varying types, and an array of removable backup media (like tape drives). The second, named backuppc, assumes that the backup media is a hard disk drive, and that the installation includes portable systems (laptops) which tend to be unavailable (off the network) on a regular basis. In terms of function, both of these systems offer the required features, but they differ in the emphasis given to different features.

The bacula system is representative of the type of application institutions have tended to use, and my example installation is indeed being used in a university environment to backup workstations and database servers. Bacula assumes the use of removable media storage, though my example installation does not employ them due to the considerable increase in the hardware costs where removable media is employed. Of course, utilization of removable media also increases the security because removable media simplify the process of off-site storage of backups. In some environments, state law or institutional policy will require off-site storage capability as part of their disaster recovery strategy. In such a case, you will probably prefer a system like bacula, complete with its database of meta-data, ability to build bare-metal recovery media, and facilities for rebuilding meta-data from scanning backup volumes. All these features come at a cost in terms of complexity of installation, required hardware and trained personnel for operation.

The backuppc program is designed to use hard disk storage as a backup media, and does not require that additional software be installed on the client (but there is often some client-side setup required if security matters -- in other words, almost always). While the setup of backuppc is much easier than in the case of bacula, it does require that a web server be run on the backup server; this is a security issue when restricting data access is a concern because it adds another major software subsystem to the server with its own security issues. Rather than use the more common volume approach (designed specifically with the storage characteristics of magnetic tape in mind), backuppc uses the disk file system image (ie the way files were originally stored on disk) as a storage format with a clever mechanism to avoid wasting space making multiple copies of the same file, and therefore maximize the utilization of available space. While backuppc can write to removable media (such as DVD), this is clearly not the focus of the program.

In summary, either of these systems can provide automated backup services, and the choice between them should reflect the requirements of your specific situation. The bacula system is more difficult to understand and setup, and requires that you install bacula client software on the client systems. Backuppc is easier to setup (everything done through a web interface), does not require special client-side software, and optimizes the utilization of disk space. Bacula is your choice if you want to backup directly to tape rather than rely on a large pool of disk space, and it does not require a web server for operation. Conversely, backuppc works well where disk is your backup media, and web services provide great utility, especially when you need users to be able to restore their own files.

Backup basics:

The perfect backup strategy is to store every bit of data that is generated on or downloaded to a given computer the instant it appears in perpetuity. This is rather like saying the way to determine the average number of leaves on a tree is to count all the leaves and divide by the number of trees. The strategy is precise but unimplementable. We must accept that an implementable backup strategy will be a compromise that recognizes the constraints we must live with in terms of hardware, software and time. Making backups is not generally considered a productive activity, but rather overhead, and therefore must be balanced against more productive activities.

The process of making a backup copy of every file on a computer system is hardware intensive, and can be expected to negatively impact the interactive performance of the client. In addition, because the backup device is connected over a network, there will also be performance implications to anyone using the network during the period of the backup. In order to minimize these performance degradations, different strategies concerning which files to backup are employed along with the selection of time slots during normally low user activity in order to minimize resource contention.

There are many files provided as part of the operating system which never change, others which change rarely, and some, such as log files, that change frequently but contain no vital information, and therefore do not need to be backed up. Recognizing this situation, it is common to adopt a methodology that does not attempt to make a copy of every file on the client with each backup run. Commonly, system files are not backed up at all, because in the case of catastrophic hardware failure, the system will be reinstalled (there must be an installed operating system to be able to communicate over the network, so why backup the system files to a network backup device?). The first decision, therefore, is which files should be backed up over the network? This will vary for different types of systems, but for workstations will generally be the user area (/home for instance). Servers will probably add the area where the data they serve is stored (such as /var/www for web servers). Clearly, it is important to communicate to your clients that all files are not being backed up, and where they must store files to ensure they will be placed on backups.

After the decision of what to back up comes the discussion of how often to make a backup run. Given the goal to minimize negative performance impacts on systems, and knowledge of the varying rate of change for files, different types of backup runs are generally made. A run which copies every file located in the area for backups is named a full backup (notice from the last paragraph that this generally does not mean every file on the system). A full backup is the most hardware and bandwidth intensive type of backup run, and will almost invariably make copies of many files which have not changed since the last full backup. Taking advantage of this realization, one could backup only the files that have changed since the last full backup. This type of differential backup would require the existence of a full backup to be useful for restores because it would contain far fewer files (none of the ones that have not changed), so would also have far lower performance impact on the client and network. This meets the goal of lowering the impact of backups, but over time the differential backup size would grow as changes and data accumulate; in fact, each successive differential backup would be larger than

the last. To lower impact further, one could backup only the files which have changed since the last backup run of any type. If backup runs were made daily, this incremental backup would only ever contain the files changed or added in the last day.

Each of the types of backups exhibits trade-offs that need to be considered. A full backup has all the files, changed or not, and is all that is needed to restore the backed up area to the same state it was during the full backup. It is, unfortunately, the most hardware and bandwidth intensive type, so while valuable, it is intrusive. A differential backup contains all the files which have changed or been added since the full backup on which it was based. This can save lots of work if the systems are relatively static, but requires the full backup as well for a complete restoration of all files. Incremental backups exhibit minimum impact, but are based on multiple other backups, which may include differential as well as other incremental backups. This can make restoration a somewhat tedious and time-consuming process.

Comparing the performance impact with the normal schedule of operations in your specific installation will provide a minimum impact solution. For instance, you might schedule a periodic full backup over a weekend (assuming minimum activity on the weekend). In addition to the performance impact, full backups require more time to complete, so weekends are generally a good time for full backups. Once you have a full backup of a system, that can be leveraged by running differential backups against the full backup. On relatively static systems, this can avoid the impact of full backups by doing differentials on several weekends following the full backup. Incremental backups can be done each evening against the backup of the previous day, regardless of its type, and because they are low impact, can usually complete during in the wee hours of the morning or sometimes even during the workday. This backup schedule, and what it implies for restores, needs to be an integral part of your infrastructure documentation which is available to all interested or affected parties.

Bacula backup program:

The bacula web presence is at www.bacula.org where you will find documentation and an active development community. If you already have experience with enterprise level backup systems (tape drives or silos), the terms and software will probably seem familiar to you. If you are new to this application area, the amount of information and terminology can seem daunting. The approach used in bacula assumes that you are using removable backup media (tapes of some sort) which are rotated over time and reused on a scheduled basis. While bacula works perfectly well with disk-based volumes instead of tape, some of the discussions in the documentation will seem unusual to you if you do not think in terms of removable media to which data is streamed, rather than a file system on a random access disk device.

The services bacula provides are the result of communication between a group of daemon programs rather than one monolithic server. It is important from the beginning to understand what programs must run and which ones communicate with each other to accomplish a given task. Take the time to read the introductory material to gain an appreciation for the paths of communication between processes noting that while these communication paths can all be local (between processes in the same machine), in a typical installation they will span a network. In addition, bacula depends on an external database system to keep track of which files are backed up, and on which volume. Remember that tape is not a random access device,

and you cannot use the file system path to locate a given file; a tape must be sequentially from the beginning to find the file, so it is important to know in advance on the tape (volume) the file resides.

The first step then is to address this dependency by installing the database system you prefer to use with bacula. There is support for MySQL, PostgreSQL and SQLite in bacula with discussions in the documentation about issues between the three. Because the project this example installation serves had selected PostgreSQL for the data storage (on a different machine, not the bacula server), I decided to use PostgreSQL as well to try to minimize the number of different tools involved in the infrastructure. After you have installed the database system and can add users, make a bacula user and work through the section entitled “A Brief Tutorial” on the bacula web site. This will acquaint you with the different steps involved in making and restoring backups, and test both the database and the bacula server.

Bacula and firewalls:

One issue that is not discussed in detail in the bacula documentation is how to deal with firewalls. Certainly, the backup server should run a firewall that limits access to all functionality other than ssh and bacula services, and even those should be limited to the systems that are actually being backed up. By default, bacula uses the following service addresses (these lines can be added to the /etc/services file if they do not already exist):

```
bacula-dir      9101/tcp
bacula-fd       9102/tcp
bacula-sd       9103/tcp
```

The server will need an iptables rule similar to this for each client:

```
iptables -A INPUT -i eth0 -p tcp -s $CLIENT -d $PUBLIC --dport 91 03 -j ACCEPT
```

In this rule, the CLIENT variable should be the IP address of the client, and PUBLIC the address of eth0 on the network. Each client will be represented by a rule of this type on the server which allows them access to the bacula storage daemon. In addition, if you want to be able to run the console applications from other systems in addition to the server, they will need a rule of the following form:

```
iptables -A INPUT -i eth0 -p tcp -s $CLIENT -d $PUBLIC --dport 9 101 -j ACCEPT
```

This rule allows client applications to access the bacula director service. Each client will need to allow access from the bacula server to their bacula file daemon:

```
iptables -A INPUT -i eth0 -p tcp -s $BACULA --dport 9102 -j ACCEPT
```

In this example rule, the BACULA variable must be the IP address of the bacula server. Once the rules are in place, you should have allowed access between the various client and server daemons involved in the bacula system. Once you have successfully backed up a remote client, it is time to begin planning your actual installation.

Bacula configuration:

After having installed and tested bacula, you are now ready to configure to meet the needs of your site. The following examples are taken from a working installation and reflect common requirements and compromises. My example project collects samples from environmental instruments and stores them in a database where they are queried by a web server to generate reports showing historical trends. The problem of project backups focuses on the database where all the samples are stored as well as some workstations where software and/or documentation is developed, and some “data aggregation” systems where raw data records are stored until processed (in this case converted to sql insert statements for insertion in the central database). In this case, all the systems are Linux based, so the same client software can be used for all systems, but there are bacula clients for Macs and Windows OS versions as well. In addition, this installation uses a redundant RAID disk system as the backup device because removable media devices, especially ones that can support automated backup procedures, are rather expensive.

Customizing the default bacula installation for your needs is accomplished by modifying some of the configuration files. You will start with the existing tutorial versions, and modify them to more directly reflect your site. For instance, my example installation uses a RAID disk array for backup storage volumes. I needed three classes of backups (desktops, database servers, and a compute cluster controller) and the ability to make DVD archive copies for carrying work off-site. I expect to maintain the system by running console programs from my desktop workstation which happens to be in a different building from the storage array. All of these requirements must be reflected in the configuration files.

The different type of systems to be backed up are described in JobDefs records in the bacula-dir.conf file. Here is an example JobDefs record:

```
JobDefs {
  Name = DesktopJob
  Type = Backup
  Level = Full
  Client = bacula-fd
  FileSet = DesktopFileSet
  Schedule = WeeklyCycle
  Storage = DesktopFileStorage
  Messages = Standard
  Pool = Desktops
  Priority = 10
}
```

The JobDefs record is a template which describes defaults for clients of this type. The fields can be over-ridden for each specific client in its Job record when the default is not appropriate. The values being assigned in this record are in most cases names of other record types which carry more specific information. For instance, the Schedule in the above JobDefs record is assigned as WeeklyCycle. That means to expect in the same configuration file a record of type Schedule with the name of WeeklyCycle which defines the schedule of backups for the DesktopJob class of backups, and here is what it might look like:

```
Schedule {
  Name = WeeklyCycle
  Run = Full 1st sun at 23:05
  Run = Differential 2nd-5th sun at 23:05
  Run = Incremental mon-sat at 23:05
}
```

This schedule will begin full backups at 23:05 on the first Sunday of each month, differential backups at the same time on all other Sundays in the month, and incremental on each other day of the month. Of particular interest in customization of an installation are the FileSet, Storage and Pool definitions for this class. To find the set of files normally backed up for the DesktopJob class, the FileSet record type named DesktopFileSet should be examined:

```
FileSet {
  Name = DesktopFileSet
  Include {
    Options {
      signature = MD5
      compression=GZIP
    }
    File = /home
  }
}
```

The File value shows that all files under the directory /home will be in the backup set for hosts of this class. Any file stored in other locations on DesktopJob clients will not be in the backups unless we override this value with a different FileSet setting for a given client. The options show that we want the stored files to be compressed in order to save disk space.

The JobDefs record above refers to a Storage type record named DesktopFileStorage that provides more details about communicating with the storage daemon to be used for this class:

```
Storage {
  Name = DesktopFileStorage
  Address = bacula.mydomain.edu
  SDPort = 9103
  Password = "very_long_random_storage_access_password_string"
  Device = DesktopDevice
  Media Type = DesktopMedia
}
```

This record shows the system name and port for the storage daemon which will provide for the backup storage used by this class of clients. Notice that there is a Password field which shows the necessary authentication information the client will be expected to provide before the storage daemon will actually proceed with a connection request. The Device and Media Type must be reflected by records in the bacula-sd.conf file where information on the actual storage mechanisms will be detailed:

```
Device {
  Name = DesktopDevice
  Media Type = DesktopMedia
  Archive Device = /backups/desktops
```

```

LabelMedia = yes;
Random Access = Yes;
AutomaticMount = yes;
RemovableMedia = no;
AlwaysOpen = no;
}

```

The Device record above which was named in the bacula-dir.conf file shows that the Archive Device is actually a mounted file system (the mount point is /backups/desktops) as well as other useful information including the Media Type. Finally, the JobDefs record refers to a Pool record type named Desktops :

```

Pool {
  Name = Desktops
  Pool Type = Backup
  Maximum Volume Bytes = 4g
  LabelFormat = "DSK-"
  Volume Retention = 30 days
  Recycle = yes
  RecyclePool = Desktops
  AutoPrune = yes
}

```

Notice that the Desktops pool contains descriptions of the storage volumes being used as 4 gigabytes in size labeled with names that begin with "DSK-" which are recycled after 30 days. All this information has described how and where storage has been allocated for backups of clients that are in the class DesktopJob, but no specific client has been named. In order to add clients using this class, Job records are defined in the bacula-dir.conf file:

```

Job {
  Name = mybox
  JobDefs = DesktopJob
  Client = mybox-fd
  Enabled = yes
  Write Bootstrap = "/home/bacula/working/mybox.bsr"
}

```

This Job record defines a backup job named mybox which uses the default information we have setup for the job class Desktopjob. The job is enabled to run with bootstrap information saved to a local file. There should be a Client record for this job named mybox-fd that would look like:

```

Client {
  Name = mybox-fd
  Address = mybox.mydomain.edu
  FDPport = 9102
  Catalog = MyCatalog
  Password = "very_long_random_client_access_password_string" # for FileDaemon
  File Retention = 30 days
  Job Retention = 6 months
  AutoPrune = yes
}

```

Each client added to the backup rotation for this class would need these two records added to the `bacula-dir.conf` file. The Password in this record allows the bacula director daemon to communicate with the client file daemon, so must be the same as the Password in the Director record in the client's `bacula-fd.conf` file.

The configuration steps above are an example of the kinds of customizations you will probably want to do to the default bacula install for your implementation. Adding clients and different classes primarily consist of making copies of record types that already exist, and modifying them to meet the new requirements. There are many options for backup devices, and it is easy to restore to a local scratch area and burn DVDs for portability. If you intend to use a disk array for backup purposes, you should consider using Linux Volume Manager to control the storage area so you can add devices in order to grow space as needed. Also remember that databases in general require special treatment in terms of backups. In general, you need to do an ASCII dump of that database, and backup that file rather than try to backup the live database. There will be information on this topic in the database documentation. This is true for the database that bacula uses as it is for the database that your project uses.

Backups and restores:

Having done all the configuration, the actual process of running backup or restore jobs is rather simple. The `bconsole` program connects to the `bacula-dir` daemon to enable issuing commands. If you enter the command `run` at the asterisk prompt, a numbered list of defined jobs will be displayed, and you can start one by entering the number (entering a "." will abort the command). There are many commands which `bconsole` can respond to which show status, manage volumes, list available resource and restore files. The tutorial is the best way to learn the most important commands, but one special case of a restore command bears mentioning. In order to restore to archive media such as a DVD or CD, it is possible to change the restore client to a local file system. That allows dumping the restore on the server machine where a command like `growisofs` can be used to burn it to DVD.

The complete set of bacula configuration files for this project are part of the addendum material for this paper. Comparing them to the tutorial configuration files, you should give you a good start on how to implement your backup service using bacula.

Backuppc program:

The home site for backuppc is <http://backuppc.sourceforge.net> where you will also find an active community and useful documentation. One of the major differences between bacula and backuppc is that backuppc is offered as a "web service", which is to say all interaction with the service is via a web interface. While this type of interface has great advantages in terms of familiarity and ease of use, you must install and manage a web server to enable backuppc. In a sense, the ease of configuration in backuppc is due to the fact that some important issues (for instance authentication) have been moved into the realm of web server configuration.

The main dependency to be met before installing backuppc is a working web server for which `apache2` serves well. You should, of course, provide firewall rules which restrict access to this web server to the actual clients being served. Remember this same machine is the backup

server and contains all the data from all the systems it serves. While it is tempting to add backuppc to your existing web service, this has unpleasant security implications and should be avoided. It is critical that you use a separate machine and restricted access web service for backuppc; if someone breaks into this machine, they have broken into every machine it serves to backup!

There may be additional dependencies which arise from your method of communication with the clients. If your clients are all Linux/UNIX systems, the best method for network backups is to use rsync on the clients because it is the most efficient with network bandwidth. You can, however, also use methods like NFS or Samba to access remote file systems, and will find examples of those types of installations in the documentation. While there is no specific client software to install on the client side, it is nonetheless necessary to do some client-side setup to ensure secure connectivity.

Because you will be using disk space for the backup pool, it would be a good idea to use LVM (Linux Volume Manager) on the disk partition used by backuppc so that you have the flexibility of adding more devices and resizing the file system in order to grow more space later. This would also be a great application for a redundant RAID-based file system. This reduces the probability of losing all backups when a disk drive fails (which will certainly happen sooner or later).

Backuppc installation:

On a fresh install of many modern Linux distributions (I used Ubuntu), the backuppc package will be immediately available for installation; simply request the installation of backuppc using the distribution's native package manager, and all other dependencies, including the apache web server, will be installed at the same time. While very desirable in terms of the small amount of effort required, this ease of installation comes at the cost of learning very little about how the installation is setup. The fact that many projects proceed without backup services of any kind argues, however, that we offer a backuppc installation as an example of a minimum learning-curve solution. This was also the reason for the selection of the Ubuntu Linux distribution for this example. Given the need for a tool which can be employed by small projects which do not have the budget for a separate system administrator, this installation discussion will be more along the line of "what to type" rather than what needs to be done. If you use different distributions, there might be more or different steps, but hopefully this walk-through will still be useful.

During the installation process of backuppc you should be informed of the password to use for accessing the backuppc service and how to change it. It is important to make a copy of that password as you will need it to configure the service later. Upon completion of the installation, there will be a web server running on the host which you can access with a browser from any networked machine. The backuppc system will be found at <http://localhost/backuppc> from the backup server machine. You should also be aware that at this point, any system on the Internet can access this web server at the address <http://your.host.addr/> and they will see the apache installation success screen.

There are two things that need to be done immediately to better secure the system. The first is to install a firewall rule so you can block access to the web server from sites that are not

backup clients. The second is to pass any site accessing the top level address of the web server (<http://your.host.name/>) directly to the backuppc service, because there should be no other reason to access this web site. Assuming a firewall that defaults to denying all inputs, a rule which allows access to the web service would look like:

```
iptables -A INPUT -i $IF -p tcp -s $CLIENT-d $PUBLIC_IP --dport 80 -j ACCEPT
```

The \$IF variable should be set to the public network interface (eth0 for instance), the \$CLIENT variable should be the IP address of a client, and the \$PUBLIC_IP variable is the IP address of the backup system. You will need a similar rule for each client, although in some instances it will make sense to allow access from an entire sub-domain instead of a per-client basis. The second issue mentioned above can be addressed by changing the index.html file in /var/www on the backup server to read:

```
<html>
<head>
<meta http-equiv="REFRESH" content="0; URL=/backuppc/">
</head>
</html>
```

This will pass clients directly to the backuppc authentication sessions without them having to remember to add the backuppc element to the host URL. These steps should not be construed as a security audit on your web service. If your institution provides such a service, you should certainly take advantage of it; if not, be vigilant about applying updates (relatively easy with Ubuntu and most main-stream Linux distributions), and limit access to as few hosts as possible. Web services are among the most common to attack for unauthorized access.

At this point, you may begin the configuration of backuppc. In order to begin, point a web browser to the <http://localhost/backuppc/> URL and you should see an authentication screen. After you login as user backuppc using the password given to you during the installation step, you will see the status screen. This will serve as the information page in the future that gives the status of the system and a brief history of recent events. From this page you can proceed to configure system-wide settings or edit configurations for an individual client. This page also has links to online documentation and a FAQ, as well as access to log files and recent email notifications.

Configure a backuppc server:

There are two approaches to configuring a backuppc server: 1) you may edit the configuration files with a text editor if you are familiar with that process 2) you may use the web interface to enter the same information. In either case, there is online documentation and the variables you are setting are represented as links to explanations of their meaning. This does not mean that you do not need to understand anything about backups to complete the configuration, but it does mean that you can do something useful without previous experience and improve the setup as you learn more, or require a more customized solution.

In keeping with the goal of this example, we will proceed down the path with the minimum learning curve. From the backuppc status screen, click the Host Summary selection on the left.

This screen will show all the clients by host name as they are configured, but initially shows only the localhost entry. You will see that there are no running jobs nor failures that need your attention. Next select the Edit Config entry. Under the Main Configuration Editor title, you will see an inactive button named Save. When you make a change on this screen, this button will become active (red), and you must click on it before leaving the screen to update the configuration file to reflect the changes you have made. If you forget to do this, no changes will be made to the configuration file.

Under the Save button you will see a list of seven headings which represent the areas that can be configured with the interface. You will not necessarily need to configure every item under each heading as the defaults are often adequate. All configuration screens offer the same interface, but for different variables. On the left you will see the variable name in blue. If you click on the name, you will be shown an explanation of what that variable controls. This explanation is not necessarily exhaustive, and you may need to read more documentation if you are confused, but it is a reminder of what you are configuring. To the right of the variable will be a text box in which you can edit, or, in some cases, a list from which you can select items. In addition, there will sometimes be command buttons like Add, Insert or Delete. Pressing these buttons which cause an action such as create a new text box for entry of information or delete values from the text box to the immediate right of the button. In some cases, there are also some instructions printed in a text box at the bottom (for example under the Hosts heading).

Start by selecting the Hosts heading. In this configuration screen you add the host names of client machines you want to back up. Click the Add button and a new line will appear under the localhost line but with empty text boxes. Enter the name of the host in the empty text box under the host column, and the name of the primary user in the text box under the user column (this person will receive email if there are issues with backuppc on that host). This is generally all that is needed for Linux hosts. Next select the Xfer heading to configure how the file transfer from client to server will take place. The first item is XferMethod which allows you to choose between several methods for doing the actual file transfer. Click on the arrow to drop down the selection list, and click on rsync. This utility is probably already on your Linux/UNIX host, but if not should be easy to install. Other methods will be more useful for different operating systems, but rsync is the most efficient with network bandwidth, which must always be a consideration in backup procedures. Farther down in the Include/Exclude section you will find BackupFilesOnly. Enter the path /home into the text box and click Add. This will build a new box with a Delete button so you can later delete that path if you decide not to include all the files under the /home directory in your backups. It will rarely be the case that you will want to backup all files for reasons mentioned in the Backup Basics section above. Under the Rsync Paths/Commands/Args section you will notice variables for the RsyncClientCmd. This entry may not be intelligible to you, but it implies that you can ssh and rsync command to the client machine as root. That is an unacceptable security risk, so we will need to come back to this item later and modify it to arrive at an acceptably secure implementation.

You can take a few moments at this point to look at the other headings and variables, but the items configured up to this point will be enough to get a basic backup started. What you have been editing are site-wide configurations that work the same for each client. Once clients are

added, however, you can select them from the Host Summary screen and make configuration alterations specific to individual clients (for instance add or modify the backup paths) which will over-ride the settings we have just made in the site configuration file. Don't forget to click the Save button so all these changes will be written back to the system configuration file.

Backuppc client setup:

While there is no client-side software to install, it is necessary to be able to access the area where files on a client are stored, and to move them onto the backuppc system. The preferred utility to do this on Linux/UNIX systems is rsync; if you use one of the other provided access methods, your setup will be different from this description. The main issue in setting up rsync is how to securely execute the necessary command on a remote system with sufficient permissions to allow the desired files to be copied to the server, yet not leave the client vulnerable to having files copied by unauthorized users or to other systems. The backuppc FAQ mentions multiple solutions to this problem, so you should check there if the following solution is too confusing to you.

The client setup for rsync use is a two stage process. The first is to allow an unprivileged user named backuppc access to all files on the system (inaccessible files cannot be backed up). The second step is to allow access to the system as that user from a remote system. As you can guess, allowing any user access to all files is a security problem before you even think of allowing that access from remote systems! This process must be done very carefully to ensure you have not subverted the normal security mechanisms of the operating system.

Step one is to make a user account for backuppc on both the server and all client systems, complete with passwords of course. These backuppc accounts on all systems should be normal users with no special privileges. On each client, we will add the privilege for the backuppc user to access all files, but in a very restricted manner. The sudo command allows a normal user elevated privileges as long as the user is listed in the sudoers file and only for the command also shown in the file. An entry in the client /etc/sudoers file which will allow the backuppc user to execute the necessary command as root would look something like:

```
backuppc ALL=NOPASSWD: /usr/bin/rsync --server --sender *
```

This allows the backuppc user to execute an rsync command of that form with adequate permissions to guarantee that it can read all files and enter all directories necessary. Now the local users backuppc has the necessary permissions, but for this to be useful for backuppc purposes, the command must be issued remotely from the server.

Step two is in order to allow execution of the sudo command over an ssh link that is both secure and encrypted. The backuppc user on the server be able to ssh a command to the client (the rsync command) without having to authenticate. To accomplish this, on the server use the `su - backuppc` (substitute user backuppc, and yes the "-" character is required) to "become" the backuppc user. As user backuppc, generate a valid ssh authentication key using the command `ssh-keygen -t dsa` and accept the defaults; when asked for a pass phrase, leave it empty (ie type just the Enter key) both times. This will generate ssh authentication keys in the directory named `.ssh` located in the backuppc user's home directory. Now `cd ~/.ssh` in order to

change directory to the location where ssh keys are stored, and you will see a group of files, one of which is named `id_dsa.pub`. That is the public portion of the backuppc user key, and it must be copied to each client's backuppc user directory (`scp id_dsa.pub client.dom.edu:`). You should be asked for the password for user backuppc on the client system while doing this. If not, do not proceed because something is amiss (reread the section above carefully). When you supply the password for the user backuppc on the remote client system, the copy should succeed. At that point use ssh to login to the client (`ssh client.dom.edu`) and again supply the password for that system. You should find the `id_dsa.pub` file you just copied to the system. In order to place the key where it will be found when you ssh to the client, move it to the client's ssh key storage directory renaming the file (`mv id_dsa.pub ~/.ssh/authorized_keys`). At this point, the client's ssh program should recognize the server system and have access to the public part of backuppc user's authentication key. To test this, logout of the client system (`exit`) and use the same ssh command as above to access the system again. This time, the backuppc user should not be prompted for a password because the authentication can be accomplished using the public key you previously stored on the system.

It is important to understand the security implications of what has been done to this point. The client system has been modified to allow one user, backuppc, access all files on the system with the `rsync` command (but no other). The backuppc user on the server has placed the public portion of their ssh authentication key in the authorized keys file of the backuppc user on the client, so any command the backuppc user on that client can successfully issue, can also be issued from the server without the need for authentication (ssh has already supplied the authentication). Anyone who can successfully login as the backuppc user on the client, can also issue the `rsync` command (or any other command that backuppc can use), but unless they are coming from the server, they will have to authenticate (supply the password for the backuppc user on the client). This protection is only valid as long as the private portion of the ssh authentication key, named `id_dsa`, is kept private (in other words, it should be stored only on the server system). An additional measure of security would be to allow ssh access to the client only from the backuppc server so that even if the private part of the ssh authentication key were stolen, the firewall would not allow access from other machines which might try to use the stolen key. This is simply done by adding a firewall rule of the form:

```
iptables -A INPUT -i $IF -p tcp --syn -s $SRV -d $PUB_IP --dport 22 -j ACCEPT
```

In this rule, the `$IF` should be the network interface name (such as `eth0`), the `$SRV` should be the IP address of the backuppc server and the `$PUB_IP` should be the IP address of the client. Use of this rule assumes that the default for the `INPUT` chain is `drop` so that ports must be specifically opened by a rule such as the above. I give an example of such a firewall script in the addendum.

Having made the changes above to the client, it will be necessary to modify the backuppc configuration file to reflect the sudo command setup that was made on the client as was mentioned above. On the backuppc server, go back to the Edit Config screen and select the Xfer Method object. Because we previously selected the `rsync` method, you should find a variable setting named `RsyncClientCmd` that contains this command line:

```
sshPath -q -x -l root $host $rsyncPath $argList+
```

This command attempts to ssh to the client system as the user root which we need to change so that the login user is backuppc and the command is run under sudo control (the changes we edited into the /etc/sudoers file on the client). The correct command now would be:

```
$sshPath -q -x -l backuppc $host sudo $rsyncPath $argList+
```

By comparing the two versions of the command, you will see that we have avoided leaving the client accessible to root without a password by making the backuppc user and giving them the specific privileges required and no more. The same modification will be necessary to the command named RsyncClientRestoreCmd so the files can be written back correctly.

You now have the first remote client added to backuppc. Adding additional clients will only require a new host record (select the Hosts tab on the Edit Config screen) as long as the backup paths and other settings are the same as the first host. You must add the backuppc user on the new host, make the addition to the /etc/sudoers file, and copy over the backuppc user id_dsa.pub file from the server exactly as was done for the first host. Before adding another host, however, it would be a good idea to check that the first host backs up correctly.

Testing backuppc:

The best test of the backuppc installation is, of course, to backup a host, then see if you can restore some files. It will help, however, to make a few preliminary tests first. Before any form of backup can be performed, the server will need appropriate access to the client. The procedure above for moving the public part of the ssh authentication key for the backuppc user was intended to meet that requirement. To ensure the procedure was successful, on the server execute the substitute user command to become the backuppc user (su - backuppc). You should now be able to execute commands remotely on the client via ssh (a simple test might be ssh client.dom.edu date). The command should execute on the remote machine without requiring authentication (ie no prompt for password). If this command works correctly, then you can try executing a backup sequence from the command line (the advantage of testing from the command line is the you will see any error messages immediately that might give a clue to what issues remain to be resolved). To start a backup run from the command line:

```
/usr/share/backuppc/bin/BackupPC_dump -v -f $Client
```

The command above is correct for the Ubuntu installation, but it is possible your installation has used a different directory path for the install. You can find the correct directory path on the Edit Confi screen under the Server path in the item InstallDir (the bin directory will be located there, so it will be correct). This command should run a verbose backup session with the client. If there are problems, you should have information from the messages that will help you isolate the problem using the documentation, especially the FAQ. Assuming the happy case where this backup runs correctly, you now have a model client you can use to add other clients to the backup rotation.

Restoring files with backuppc:

In order to restore files using backuppc, select the Host Summary screen and on that screen select the the host whose files you need to restore. This will take you to a Host Backup Summary screen for that host which displays the lists of backups done for that host along with the date they were made. Selecting a backup number will take you to a Browse Backup screen which, on the left, has a familiar “tree” view of the files on the system and on the right, a list of files in the current directory with a selection box by each. At the top of the list is one box labeled Select All, for obvious purposes, and to the right of it a button labeled Restore Selected Files. As you might imagine, you locate and select the files to be restored, then click on the Restore Selected Files button to accomplish the task.

System administration details:

As you might expect there are many details that have been skimmed over or not even mentioned in this document. Among the most important for both system is the installation and maintenance of a good firewall on the backup system. Rules have been given above as examples to illustrate what form a useful rule will take. These rules assume the installation of a basic firewall on both server and clients that denies access to other hosts as a default, and must have added rules to open allowed access. I have appended an example of such a firewall in the addendum as a starting point, but this is the single most important security topic for network attached hosts, and you are encouraged to get local help in arriving at a good firewall.

There may also be additional steps required to ensure all the daemon processes are restarted correctly after reboots. In this case, backuppc should only requires checking on the server as there are no daemons running on the clients. Users of bacula also must ensure the bacula-fd on the client is running correctly after client reboots.

In some cases, specific commands will need to be run before you can successfully make a backup copy of a file. The most notorious example of this is the database file. Because many databases (such as PostgreSQL and MySQL) use “sparse” files for storage which cannot be successfully backed up by simply making a copy of the file. In these cases you must arrange for a utility (such as pg_dump in PostgreSQL) to be run before the backup which will leave an ASCII copy of the data in a dump file. This ASCII copy is actually a series of commands and data that will rebuild the database if used as input to the database system. For this reason, you will often find “pre” and “post” commands in backup scripts. These allow for special actions to be taken in the cases where files which are likely to change during a backup run can be treated with special utilities designed to “snapshot” the file.

Addendum:

Here you will find some hopefully useful example files that were deemed an interruption of the topic above, and so promised at the end of the document. First is an example firewall for a Linux system that can serve (with modifications) both servers and clients. Be aware that this firewall allows neither email connections nor pinging of the host. It is intended to be install as and init script in /etc/init.d where it will automatically be run during boot on a Debian style system.

```
#!/bin/bash

# iptables rules for Linux workstations
#

case "$1" in
  start)
    echo "Enabling iptables"
    DEFAULT_INPUT="DROP"
    ;;
  stop)
    echo "Enabling iptables (open)"
    DEFAULT_INPUT="ACCEPT"
    ;;
  *)
    echo "Usage: /etc/init.d/site.iptables {start|stop}"
    exit 0
    ;;
esac

#-----
# variables

IF="eth0"
IP=`ifconfig eth0 | grep 'inet addr' | awk '{ print $2 }' | awk 'BEGIN {FS=":"}
{ print $2 }'`

if [ -z "${IP}" ] ; then
    echo "Error trying to determine eth0 ip address"
    exit
fi

IPTABLES="/sbin/iptables"
# networks
ANY="0/0"
MYNET="192.168.1.0/24"
# hosts
SERVER="192.168.1.100"
CLIENT1="192.168.1.101"
CLIENT2="192.168.1.102"

#-----
# basic setup

# Remove all existing rules belonging to this filter
```

```
/sbin/iptables -F INPUT
/sbin/iptables -F OUTPUT
/sbin/iptables -F FORWARD

# Enable IP spoofing protection (Source Address Verification)
for f in /proc/sys/net/ipv4/conf/*/rp_filter; do
    echo 1 > $f
done

# Disable ICMP Redirect Acceptance
for f in /proc/sys/net/ipv4/conf/*/accept_redirects; do
    echo 0 > $f
done

# Disable Source Routed Packets
for f in /proc/sys/net/ipv4/conf/*/accept_source_route; do
    echo 0 > $f
done

# Ignore ICMP echo requests on broadcast and multicast addresses
echo "1" > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts

# Set the default policies for the packet filter
$IPTABLES -P INPUT ${DEFAULT_INPUT}      # reject input
$IPTABLES -P OUTPUT ACCEPT              # unrestricted output
$IPTABLES -P FORWARD DROP              # no forwarding

# turning firewall off
if [ "${DEFAULT_INPUT}" = "ACCEPT" ] ; then
    exit
fi

# Unlimited traffic on the loopback interface.
$IPTABLES -A INPUT -i lo -j ACCEPT
$IPTABLES -A OUTPUT -o lo -j ACCEPT

# allow udp from network
$IPTABLES -A INPUT -i $IF -p udp -d $IP -j ACCEPT

# allow traffic to established connections only (! --syn)
$IPTABLES -A INPUT -i $IF -p tcp ! --syn -j ACCEPT

# uncomment to allow SSH server (22) connection requests from clients
#$IPTABLES -A INPUT -i $IF -p tcp --syn -s $CLIENT1 -d $IP --dport 22 -j ACCEPT

# uncomment to allow web service access
#$IPTABLES -A INPUT -i $IF -p tcp -d $IP --dport 80 -j ACCEPT
```

Here you will find example configuration files for the bacula system. There are two that are necessary only on the server: bacula-dir.conf and bacula-sd.conf. One additional file named bacula-fd.conf is required on each client (including the server if it is to be backed up as well). There is also a bconsole.conf file which must be present on all system from which you wish to connect to the bacula dir daemon. Here is an example of a complete bacula-dir.conf file:

```
# Global directives
Director {
  Name = bacula-dir
  DIRport = 9101
  QueryFile = "/home/bacula/conf/query.sql"
  WorkingDirectory = "/home/bacula/working"
  PidDirectory = "/home/bacula/var"
  Maximum Concurrent Jobs = 1
  Password = "very_long_random_console_access_password_string" # Console password
  Messages = Daemon
}

# When to do the backups, full backup on first sunday of the month,
# differential (i.e. incremental since full) every other sunday,
# and incremental backups other days
Schedule {
  Name = WeeklyCycle
  Run = Full 1st sun at 23:05
  Run = Differential 2nd-5th sun at 23:05
  Run = Incremental mon-sat at 23:05
}

# Generic catalog service
Catalog {
  Name = MyCatalog
  dbname = bacula; user = bacula; password = ""
}

# message delivery -- send most everything to email address and to the console
Messages {
  Name = Standard
#
  mailcommand = "/home/bacula/bin/bsmtp -h localhost -f \"\\(Bacula\\) \\<%r\\>\" -s
  \"Bacula: %t %e of %c %l\" %r"
  operatorcommand = "/home/bacula/bin/bsmtp -h localhost -f \"\\(Bacula\\) \\<%r\\>\"
  -s \"Bacula: Intervention needed for %j\" %r"
  mail = bacula@domain.edu = all, !skipped
  operator = bacula@domain.edu = mount
  console = all, !skipped, !saved
  append = "/home/bacula/working/log" = all, !skipped
}

# Message delivery for daemon messages (no job).
Messages {
  Name = Daemon
  mailcommand = "/home/bacula/bin/bsmtp -h localhost -f \"\\(Bacula\\) \\<%r\\>\" -s
  \"Bacula daemon message\" %r"
  mail = bacula@domain.edu = all, !skipped
  console = all, !skipped, !saved
  append = "/home/bacula/working/log" = all, !skipped
}
```

```

}

# Job classes
# desktop systems
JobDefs {
    Name = DesktopJob
    Type = Backup
    Level = Full
    Client = bacula-fd
    FileSet = DesktopFileSet
    Schedule = WeeklyCycle
    Storage = DesktopFileStorage
    Messages = Standard
    Pool = Desktops
    Priority = 10
}

FileSet {
    Name = DesktopFileSet
    Include {
        Options {
            signature = MD5
            compression=GZIP
        }
        File = /home
    }
}

Storage {
    Name = DesktopFileStorage
    Address = bacula.subdomain.university.edu
    SDPort = 9103
    Password = "very_long_random_storage_access_password_string"
    Device = DesktopDevice
    Media Type = DesktopMedia
}

Pool {
    Name = Desktops
    Pool Type = Backup
    Maximum Volume Bytes = 4g
    LabelFormat = "DSK-"
    Volume Retention = 30 days
    Recycle = yes
    RecyclePool = Desktops
    AutoPrune = yes
}

# project systems
JobDefs {
    Name = TE0Job
    Type = Backup
    Level = Full
    Client = bacula-fd
    FileSet = TE0FileSet
    Schedule = WeeklyCycle
    Storage = TE0FileStorage
    Messages = Standard
}

```

```

Pool = TE0
Priority = 10
}

FileSet {
Name = TE0FileSet
Include {
Options {
signature = MD5
compression=GZIP
}
File = /home
File = /data/teo_backups/database.dmp
}
}

Storage {
Name = TE0FileStorage
Address = bacula.subdomain.university.edu      # N.B. Use a fully qualified
name here
SDPort = 9103
Password = "very_long_random_storage_access_password_string"
Device = TE0Device
Media Type = TE0Media
}

Pool {
Name = TE0
Pool Type = Backup
Maximum Volume Bytes = 4g
LabelFormat = "TE0-"
Volume Retention = 30 days
Recycle = yes
RecyclePool = TE0
AutoPrune = yes
}

# compute cluster
JobDefs {
Name = TB4Job
Type = Backup
Level = Full
Client = bacula-fd
FileSet = TB4FileSet
Schedule = WeeklyCycle
Storage = TB4FileStorage
Messages = Standard
Pool = TB4
Priority = 10
}

FileSet {
Name = TB4FileSet
Include {
Options {
signature = MD5
compression=GZIP
}
}
}

```

```

    File = /home
    File = /data/raid2
    File = /data/raid3
  }
}

Storage {
  Name = TB4FileStorage
  Address = bacula.subdomain.university.edu      # N.B. Use a fully qualified
name here
  SDPort = 9103
  Password = "very_long_random_storage_access_password_string"
  Device = TB4Device
  Media Type = TB4Media
}

Pool {
  Name = TB4
  Pool Type = Backup
  Maximum Volume Bytes = 4g
  LabelFormat = "TB4-"
  Volume Retention = 30 days
  Recycle = yes
  RecyclePool = TB4
  AutoPrune = yes
}

# Clients to backup
# client1.subdomain.university.edu
Job {
  Name = client1
  JobDefs = DesktopJob
  Client = client1-fd
  Enabled = yes
  Write Bootstrap = "/home/bacula/working/client1.bsr"
}

Client {
  Name = client1-fd
  Address = client1.subdomain.university.edu
  FDPort = 9102
  Catalog = MyCatalog
  Password = "very_long_random_client_access_password_string" # for FileDaemon
  File Retention = 30 days
  Job Retention = 6 months
  AutoPrune = yes
}

# client2.subdomain.university.edu
Job {
  Name = client2
  JobDefs = DesktopJob
  Client = client2-fd
  Enabled = no
  Write Bootstrap = "/home/bacula/working/client2.bsr"
}

Client {

```

```
Name = client2-fd
Address = client2.subdomain.university.edu
FDPort = 9102
Catalog = MyCatalog
Password = "very_long_random_client_access_password_string" # for FileDaemon
File Retention = 30 days
Job Retention = 6 months
AutoPrune = yes
}

# client3.subdomain.university.edu
Job {
    Name = client3
    JobDefs = TE0Job
    Client = client3-fd
    Enabled = yes
    Write Bootstrap = "/home/bacula/working/client3.bsr"
}

Client {
    Name = client3-fd
    Address = client3.subdomain.university.edu
    FDPort = 9102
    Catalog = MyCatalog
    Password = "very_long_random_client_access_password_string"
    File Retention = 30 days
    Job Retention = 6 months
    AutoPrune = yes
}

# client4.subdomain.university.edu
Job {
    Name = client4
    JobDefs = TB4Job
    Client = client4-fd
    Enabled = yes
    Write Bootstrap = "/home/bacula/working/client4.bsr"
}

Client {
    Name = client4-fd
    Address = client4.subdomain.university.edu
    FDPort = 9102
    Catalog = MyCatalog
    Password = "very_long_random_client_access_password_string"
    File Retention = 30 days
    Job Retention = 6 months
    AutoPrune = yes
}

# Restore template
Job {
    Name = "RestoreFiles"
    Type = Restore
    Client = restore-fd
    FileSet = "Full Set"
    Storage = File
    Pool = Default
}
```

```

    Messages = Standard
    Where = /backups/bacula-restores
}

FileSet {
    Name = "Full Set"
    Include {
        Options {
            signature = MD5
        }
        File = /home/bacula
    }
}

Client {
    Name = restore-fd
    Address = bacula.subdomain.university.edu
    FDPort = 9102
    Catalog = MyCatalog
    Password = "very_long_random_client_access_password_string"
    File Retention = 30 days
    Job Retention = 6 months
    AutoPrune = yes
}

Storage {
    Name = File
    Address = bacula.subdomain.university.edu           # N.B. Use a fully
qualified name here
    SDPort = 9103
    Password = "JvnRjX/DyRz9S1EDRwt0DX+UuC0YIVHdTUDQdnPLg5Sp"
    Device = FileStorage
    Media Type = File
}

Pool {
    Name = Default
    Pool Type = Backup
    Maximum Volume Bytes = 4g
    Recycle = yes
    AutoPrune = yes
    Volume Retention = 365 days
}

Pool {
    Name = Scratch
    Pool Type = Backup
}

```

To continue the example, here is a matching bacula-sd.conf file:

```
#
Storage {
  Name = bacula-sd
  SDPort = 9103                # Director's port
  WorkingDirectory = "/home/bacula/working"
  Pid Directory = "/home/bacula/var"
  Maximum Concurrent Jobs = 20
}

# List Directors who are permitted to contact Storage daemon
Director {
  Name = bacula-dir
  Password = "very_long_random_storage_access_password_string"
}

# Devices supported by this Storage daemon
Device {
  Name = FileStorage
  Media Type = File
  Archive Device = /backups/bacula-restores
  LabelMedia = yes;
  Random Access = Yes;
  AutomaticMount = yes;
  RemovableMedia = no;
  AlwaysOpen = no;
}

Device {
  Name = DesktopDevice
  Media Type = DesktopMedia
  Archive Device = /backups/desktops
  LabelMedia = yes;
  Random Access = Yes;
  AutomaticMount = yes;
  RemovableMedia = no;
  AlwaysOpen = no;
}

Device {
  Name = TE0Device
  Media Type = TE0Media
  Archive Device = /backups/teo
  LabelMedia = yes;                # lets Bacula label unlabeled media
  Random Access = Yes;
  AutomaticMount = yes;            # when device opened, read it
  RemovableMedia = no;
  AlwaysOpen = no;
}

Device {
  Name = TB4Device
  Media Type = TB4Media
  Archive Device = /backups/tb4
  LabelMedia = yes;                # lets Bacula label unlabeled media
```

```
Random Access = Yes;
AutomaticMount = yes;           # when device opened, read it
RemovableMedia = no;
AlwaysOpen = no;
}
```

```
# Send all messages to the Director,
# mount messages also are sent to the email address
#
Messages {
    Name = Standard
    director = bacula-dir = all
}
```

Here is a bacula-fd.conf file that would be used on a client:

```
# List Directors who are permitted to contact this File daemon
Director {
    Name = bacula-dir
    Password = "very_long_random_client_access_password_string"
}

# "Global" File daemon configuration specifications
FileDaemon {
    Name = client1-fd           # this is me
    FDport = 9102              # where we listen for the director
    WorkingDirectory = /home/user/bacula/working
    Pid Directory = /home/user/bacula/bin/working
    Maximum Concurrent Jobs = 20
}

# Send all messages except skipped files back to Director
Messages {
    Name = Standard
    director = bacula-dir = all, !skipped, !restored
}
```

To complete the example, here is a bconsole.conf file which would be placed on each host from which you wanted to run the bconsole command to control the bacula dir daemon:

```
#  
# Bacula User Agent (or Console) Configuration File  
#  
Director {  
  Name = bacula-dir  
  DIRport = 9101  
  address = bacula.subdomain.university.edu  
  Password = "very_long_random_console_access_password_string"  
}
```