Chapter 40 Backup and Recovery Methods - Notes

40.3 Learning Objectives:

- Identify and prioritize data that needs backup.
- Employ different kinds of backup methods, depending on the situation.
- Establish efficient backup and restore strategies.
- Use different backup utilities, such as **cpio**, **tar**, **gzip**, **bzip2**, **xz**, **dd**, **rsync**, **dump**, **restore**, and **mt**.
- Describe the two most well-known backup programs, Amanda and Bacula.

40.4 Why Backups?

Whether you are administering only one personal system or a network of many machines, system backups are very important. Some of the reasons are:

- **Data is valuable**

  On-disk data is an important work product, and is therefore a commodity that we wish to protect. Recreating lost data costs time and money. Some data may even be unique, and there may be no way to recreate it.

- **Hardware fails**

  While storage media reliability has increased, so has drive capacity. Even if the failure rate per byte decreases, unpredictable failures still occur. May be pessimistic to say there are only two kinds of drives: those that have failed, and those that will fail, but it is essentially true. Using **RAID** helps, but backups are still needed.

- **Software fails**

  No software is perfect. Bugs may destroy or corrupt data. Even stable programs long in use can have problems.

- **People make mistakes**

  Everyone has heard **OOPS!** (or something much worse) coming from the next cubicle (or from their own mouth) at one time or another. Sometimes, just a simple typing error can cause large scale destruction of files and data.

- **Malicious people can cause deliberate damage**

  Could be the canonical disgruntled employee or an external hacker with a point to make. Security concerns and backup capabilities are very strongly related.

- **Unexplained events happen**

  Files can just disappear without you know how, who, or even when it occured.

- **Rewinds can be useful**

  Sometimes, restoring to an earlier **snapshot** or all or part of the system may be required.

40.5 What Needs Backup?

Some data is critical for backup, some less critical, and some never needs saving. In priority order:

1. **Definitely**
   - Business-related date
   - System configuration files
   - User files (usually under `/home`)
2. **Maybe**
   - Spooling directories (for printing, mail, etc.)
   - Logging files (found in `/var/log`, and elsewhere)
3. **Probably not**
   - Software that can easily be re-installed; on a well-managed system, this should be almost everything
   - The `/tmp` directory, because its contents are indeed supposed to be temporary
4. **Definitely not**
   - Pseudo-filesystems such as `/proc`, `/dev`, and `/sys`
   - Any swap partitions or files

Obviously, files essential to your organization require backup. Configuration files may change frequently, and along with individual user's files, require backup as well.

Logging files can be important if you have to investigate your system's history, which can be particularly important for detecting intrusions and other security violations.

Don't have to back up anything that can easily be re-installed. Also, the **swap** partitions (or files) and `/proc` filesystems are generally not useful for necessary to backup, since the data in these areas is basically temporary (just like in the `/tmp` directory).

40.6 Backup vs. Archive

All backup media have finite lifetime before becoming unreadable. Conventional estimates are listed below:

- Magnetic tapes: 10-30 years
- CDs and DVDs: 3-10 years
- Hard Disks: 2-5 years

Lifetime is very sensitive to:

- Environmental conditions (temperature, humidity, etc.)
- Quality of media
- Haivng working software that can read data on current operating systems and hardware.

Lifetime is sufficient for backup, but nor for permanent digital archiving.

For lifetimes longer than the usual backup timescales, data can be preserved using multiple copies, plus copying over to newer media from time to time.

FOr very long times (i.e., many decades, centuries, etc.), standard methods do not work easily, as everything can go obsolete, hardware, software, and document format, media.

None of the inexpensive digital formats can actually compete with paper and film for long periods of time (if they are properly stored and continuously cared for - like wine).

This is a problem serious people think about and there should be good solutions available before all is lost.

40.7 Tape Drives

Tape drives not as common as they used to be. Relatively slow and permit only sequential access. On any modern setup, rarely used for primary backup. Sometimes used for off-site storage for archival purposes for long time reference. However, magnetic tape drives alw ays have only finite lifetime w ithout physical degradation and loss of data.

Modern tape drives usually of the LTO (Linear Tape Open) variety, w hose first versions appeared in the late 1990s as an open standards alternative; early formats w ere mostly proprietary. Early versions held up to 100 GB; new er versions can hold 2.5 TB or more in a cartridge of the same size.

Day to day backups are usually done w ith some form of NAS (Netw ork Attached Storage) or w ith cloud-based solutions, making new tape-based installations less and less attractive. How ever, they can still be found and system administrators may be required to deal w ith them.

In w hat follow s, w ill try not to focus on particular physical forms for the backup media, and w ill speak more abstractly.

40.8 Backup Methods

Should never have all backups residing in same physical location as systems being protected. Otherw ise, fire or other physical damage could lead to a total loss. In the past, this usually meant physically transporting magnetic tapes to a secure location. Today, this is more likely to mean transferring backups files over the Internet to alternative physical locations. Obviously, has to be done in secure w ay, using encryption and other security precautions as appropriate.

Several different kinds of backup methods can be used, often in concert w ith each other:

- **Full**

  Back up all files on the system.

- **Incremental**

  Backup all files that have changed since the last incremental or full backup.

- **Differential**

  Backup all files that have changed since the last full backup.

- **Multiple level incremental**

  Backup all files that have changed since the previous backup at the same or a previous level.

- **User**

  Backup only files in a specific user's directory.

40.9 Backup Strategies

Should note that backup methods useless w ithout associated **restore** methods. Have to take into account the robustness, clarity, ease of both directions w hen selecting strategies.

Simplest backup scheme: do a full backup of everything once, and then perform incremental backups of everything that subsequently changes. While full backups can take a lot of time, restoring from incremental backups can be more difficult and time consuming. Thus, can use a mix of both to optimize time and effort.

Example of one useful strategy involving tapes (can easily substitute other media in description):

1. Use tape 1 for a full backup on Friday.
2. Use tapes 2-5 for incremental backups on Monday-Thursday.
3. Use tape 6 for full backup on second Friday.
4. Use tapes 2-5 for incremental backups on second Monday-Thursday.

5. Do not overwrite tape 1 until completion of full backup on tape 6.
6. After full backup to tape 6, move tape 1 to external location for disaster recovery.
7. For next full backup (next Friday) get tape 1 and exchange for tape 6.

A good rule of thumb is to have at least two weeks of backups available.

40.10 Backup utilities

A number of programs are used for backup purposes:

- **cpio**
- **tar**
- **gzip**, **bzip2**, **xz**

  cpio and **tar** create and extract **archives** of files. The archives are often compressed with **gzip**, **bzip2**, or **xz**. The archive file may be written to disk, magnetic tape, or any other device which can hold files. Archives are very useful for transferring files from one filesystem or machine to another.

- **dd**

  This powerful utility is often used to transfer raw data between media. It can copy entire partitions or entire disks.

- **rsync**

  This powerful utility can synchronize directory subtrees or entire filesystems across a network, or between different filesystem locations on a local machine.

- **dump and restore**

  These ancient utilities are designed specifically for backups. They read from the filesystem directly (which is more efficient). However, they must be restored only on the same filesystem type that they came from. There are newer alternatives.

- **mt**

  Useful for querying and positioning tapes before performing backups and restores.

40.11 Using tar for Backups

tar is easy to use:

- When creating **tar** archive, for each directory given as argument, all files and subdirectories will be included in archive
- When restoring, reconstitutes directories as necessary
- Even has `--newer` option that allows incremental backups
- Version of **tar** used in Linux can also handle backups that do not fit on one tape or whatever device being used.

Few examples of how to use **tar** for backups:

- Create an archive using `-c` or `--create`:

```
$ tar --create --file /dev/st0 /root
$ tar -cvf /dev/st0 /root
```

Can specify a device or file with `-f` or `--file` option.

- Create w ith multi-volume option, using `-M` or `--multi-volume` if backup w on't fit on one device:

```
$ tar -cMf /dev/st0 /root
```

Will be prompted to put next tape w hen needed.

- Verify files w ith compare option, using `-d` or `--compare` :

```
$ tar --compare --verbose --file /dev/st0
$ tar -dvf /dev/st0
```

After making backup, can make sure that it is complete and correct using above verification option.

By default, **tar** w ill recursively include all subdirectories in archive.

When creating archive, **tar** prints message about removing leading slashes from absolute path name. While this allow s for restoring files anyw here, default behavior can be modified.

Most **tar** options can be given in short form w ith one dash, or long form w ith tw o: `-c` is completely equivalent to `--create`.

Note: can combine options (w hen using short notation) so no need to type every dash.

Furthermore, single-dashed **tar** options can be used w ith or w ithout dashes; i.e., `tar cvf file.tar dir1` has same result as `tar -cvf file.tar dir1` .

40.12 Using tar for Restoring Files

`-x` or `--extract` option extracts files from archive, all by default. Can narrow the file extraction list by specifying ony particular files. If directory specified, all included files and subdirectories are also extracted.

`-p` or `--same-permissions` options ensures files are restored w ith their original permissions.

`-t` or `--list` option lists, but does not extract, the files in the archive.

Examples:

- Extract from an archive:

```
$ tar --extract --same-permissions --verbose --file /dev/st0
$ tar -xpvf /dev/st0
$ tar xpvf /dev/st0
```

- Specify only specific files to restore:

```
$ tar xvf /dev/st0 somefile
```

- List the contents of a **tar** backup:

```
$ tar --list --file /dev/st0
$ tar -tf /dev/st0
```

40.13 Incremental Backups with tar

Can do incremental backup w ith **tar** using the `-N` (or the equivalent `--newer`), or the `--after-date` options. Either option requires specifying either a date or a qualified (reference) file name:

```
$ tar --create --newer '2011-12-1' -vf backup1.tar /var/tmp
$ tar --create --after-date '2011-12-1' -vzf backup1.tar /var/tmp
```

Either form creates backup archive of all files in `/var/tmp` w hich w ere modified after December 1, 2011.

Because **tar** only looks at file's date, does not consider any other changes to the file, such as permissions or file name. To include files w ith these changes in incremental backup, use **find** and create a list of files to be backed up.

Note: w hen follow ed by an option like `--newer`, must use the dash in options like `-vzf`, or **tar** w ill get confused. This kind of option specification confusion sometimes occurs w ith old UNIX utilities like **ps** and **tar** w ith complicated histories involving different families of UNIX.

40.14 Archive Compression Methods

Often desired to compress files to save disk space and/or netw ork transmission time, especially since modern machines w ill often find the compress -> transmit -> decompress cycle faster than just transmitting (or copying) an uncompressed file.

Number of commonly used compression schemes available in Linux. In order of increasing compression efficiency (w hich comes at the cost of longer compression times):

- **gzip**

  Uses Lempel-Zip Coding (LZ277) and produces `.gz` files.

- **bzip2**

  Uses Burrow -Wheeler block sorting text compression algorithm and Huffman coding, and produces `.bz2` files.

- **xz**

  Produces `.xz` files and also support legacy `.lzma` format.

Decompression times do not vary as much as compression times do. For day-to-day use on smaller files, one usually just uses **gzip**, as it is extremely fast, but, for larger files and for archiving purposes, the other tw o methods are often used. E.g., The Linux Kernel Archives now offers kernels only in the **xz** format.

The `.zip` format is rarely used in Linux, except to extract legacy archives from other operating systems.

Compression utilities very easily (and often) used in combination w ith **tar**:

```
$ tar zcvf source.tar.gz source
$ tar jcvf source.tar.bz2 source
$ tar Jcvf source.tar.xz source
```

for producing a compressed archive. Note: first command has the exact same effect as doing:

```
$ tar cvf source.tar source; gzip -v source.tar
```

but is more efficient because:

1. There is no intermediate file storage.
2. Archiving and compression happen simultaneously in the pipeline.

For decompression:

```
$ tar xzvf source.tar.gz
$ tar xjvf source.tar.bz2
$ tar xJvf source.tar.xz
```

or even simpler:

```
$ tar xvf source.tar.gz
```

as modern versions of **tar** can sense the method of compression and take care of it automatically.

Obviously, not w orth using these methods on archives w hose component files are already compressed, such as `.jpg` images, or `.pdf` files.

40.15 dd

dd: one of the original UNIX utilities, extremely versatile. Without options, does very low -level raw copying of files or even w hole disks. Capable of doing many kind of data conversions during the copy (such as changing byte order) and has many options to control offsets, number of bytes, block size, etc.

dd often used to read fixed amounts of data from special device nodes such as `/dev/zero` pr `/dev/random`. Basic syntax:

```
$ dd if=input-file of=output-file options
```

If the input or output files are not specified, the default is to use `stdin` and `stdout`. Doing:

```
$ dd --help
```

w ill yield a really long list of options, some frequently used, and some only very rarely.

40.16 dd Examples

Some examples of using **dd**:

- Create a 10MB file filled w ith zeros:

```
 $ dd if=/dev/zero of=outfile bs=1M count=10
```

- Backup an entire hard drive to another (raw copy):

```
 $ dd if=/dev/sda of=/dev/sdb
```

- Create an image of a hard disk (w hich could later be transferred to another hard disk):

```
$ dd if=/dev/sda of=sdadisk.img
```

- Backup a partition:

```
$ dd if=/dev/sda1 of=partition1.img
```

- Use **dd** in a pipeline:

```
$ dd if=ndata conv=swab count=1024 | uniq > ofile
```

40.17 rsync

rsync (**r**emote **sync**hronize): used to transfer files across netw ork (or betw een different locations on the same machine) as in:

```
$ rsync [options] source destination
```

Source and destination can take the form of `target:path` w here `target` can be in the form of `[user@]host`. `user@` part is optional and used if the remote user is different from the local user. Thus, these are all possible **rsync** commands:

```
$ rsync file.tar someone@backup.mydomain:/usr/local
$ rsync -r a-machine:/usr/local b-machine:/usr/
$ rsync -r --dry-run /usr/local /BACKUP/usr
```

Have to be very careful w ith **rsync** about exact location specifications (especially if you use the `--delete` option), so it is highly recommended to use the `--dry-run` option first, and then repeat if the projected action looks correct.

rsync is very clever; checks local files against remote files in small chunks, and very efficient in that w hen copying one directory to a similar directory, only the differences are copied over the netw ork. This synchronizes the second directory w ith the first directory. One often uses the `-r` option w hich causes **rsync** to recursively w alk dow n directory tree copying all files and directories below the one lsited as the `sourcefile`. Thus, very useful w ay to back up project directory might be similar to:

```
$ rsync -r project-X archive-machine:archives/project-X
```

Simple (and very effective and very fast) backup strategy: simply duplicate directories or partitions across a netw ork w ith **rsync** commands and do so frequently.

40.18 cpio

cpio (**co**py **i**n and **o**ut): general file archiver utility that has been around since the earliest days of UNIX, originally designed for tape backups. Even though new er archiving programs (like **tar**, w hich is not exactly young) have been deployed to do many of the tasks that w ere once in the domain of **cpio**, it still survives.

E.g., have already seen the use fo **rpm2cpio** to convert RPM packages into **cpio** archives and then extract them. Also, the Linux kernel uses a version of **cpio** internally to deal with **initramfs** and **initrd** initial ram filesystems and disks during boot. One reason **cpio** lives on is that it is lighter than **tar** and other successors, even if it is somewhat less robust.

Examples of using **cpio**:

- Create an archive, use `-o` or `--create`:

```
$ ls | cpio --create -O /dev/st0
```

- Extract from an archive, use `-i` or `--extract`:

```
$ cpio --i somefile -I /dev/st0
```

- List contents of an archive, use `-t` or `--list`:

```
$ cpio -t -I /dev/st0
```

Can specify the input (`-I` device) or use redirection on the command line.

The `-o` or `--create` option tells **cpio** to copy files out to an archive. **cpio** reads a list of file names (one per line) from standard input and writes the archives to standard output.

The `-i` or `--extract` option tells **cpio** to copy files from an archive, reading the archive from standard input. If you list files names as patterns (such as `*.c`) on the command line, only files in the archive that match the patterns are copied from the archive. If no patterns give, all files extracted.

The `-t` or `--list` options tells **cpio** to list archive contents. Adding the `-v` or `--verbose` option generates long listing.

40.19 dump and restore

dump and **restore** utilities: been around since earliest days of UNIX, not originally designed for modern hardware, filesystems, storage capabilities.

Unlike **cpio** and **tar**, these utilities directly read and write filesystem, more efficient and allows backup files to be created without affecting time stamps.

Positive features:

- Can perform full or incremental backups
- Understand specific filesystem format and hot to read and write it
- Efficient when creating full backups, head movement reduced
- Can specify output tape size and density, block size and count, or even both
- Can dump to any valid device or file; defaults to `/dev/tape`
- Parameters in `/etc/fstab` control what gets dumped and when.

Negative features:

- Multiple filesystem passes are required for backups
- Only works with ext2, ext3, and ext4 filesystems. (Other filesystems may have their own utilities, such as **xfsdump**)
- Cannot be run safely on mounted filesystems.

Lack of filesystem flexibility: rather strong limitation, both because of multiplicity of filesystem tapes in use in Linux, and because modern trend is to abstract details such as exactly how data is stored.

dump and **restore** sometimes used by higher-level backup program suites, such as Amanda. Some familiarity with these legacy tools is useful.

40.20 dump Options

dump has a number of options, including letting you set parameters. Some of these include:

- `-0-9`

  Dump level. Level 0 is full backup and higher numbers are incremental backups.

- `-B records`

  Records per volume.

- `-b blocksize`

  KB per record.

- `-f file`

  Output device or file.

- `-u`

  Update `/etc/dumpdates`

- `-w`
  Print most recent dump date of each filesystem in `/etc/dumpdates`.

(Note: some systems, such as Debian-based ones, may use `/var/lib/dumpdates` instead.)

Parameter values need not be placed immediately after the option, but must be given in the same order as the options that specify them.

40.21 restore

restore used to read archives, tapes, or files which were created by **dump**. E.g., to restore all files that were dumped, relative to current directory:

```
$ sudo restore -rvf /tmp/boot_backup
```

Useful options to **restore**:

- `-r`

  Restores everything. The dumped material is read and the complete contents are loaded into the current directory.

- `-t`

  Files and directories specified are listed on standard output if they reside on the backup. If no file argument is specified, the root directory on the backup is listed. No actual restore is performed with this option.

- `-x`

  The files and directories named are extracted from the backup. If the named file matches a directory on the backup, the directory is recursively extracted. If no argument is listed, then all contents of the backup are extracted.

- `-i`

  This mode allow s the interactive restoration of files form the backup. After reading in the directory information from the backup, **restore** provides a shell-like interface that allow s the user to move around the directory tree selecting files to be extracted.

You cannot use `-r` and `-s` at the same time.

40.22 mt

mt utility is used to control magnetic tape devices. Most often used to query or position the tape before or betw een backups and restores. By default, **mt** uses tape drive defined in `TAPE` environment variable. Can also use the `-f device` option to specify tape.

Note: only root user can use **mt**. Syntax:

```
mt [-h] [-f device] operation [count] [arguments...]
```

w here:

- `-h` : is for displaying usage (help)
- `-f device` : is for specifying the tape device
- `operation` : is one of the tape operations
- `count` : is used for some operations w hich can repeat (default is 1)
- `arguments` : are used for some operations.

Some examples of using **mt**:

- Show satus information about the tape device:

```
$ mt status
```

- Rew ind the tape:

```
$ mt rewind
```

- Erase the tape:

```
$ mt erase
```

- Move forw ard to the end of the current archive:

```
$ mt fsf
```

40.23 Backup Programs

No shortage of available backup program suites available for Linux, including proprietary applications for those supplied by storage vendors, as w ell as open-source applications. Several that are particularly w ell know n:

1. **Amanda** (**A**dvanced **M**aryland **A**utomatic **N**etw ork **D**isk **A**rchiver) uses native utilities (including **tar** and **dump**) but is far more robust and controllable. **Amanda** is generally available on Enterprise **Linux** systems through usual repositories. Complete information can be found on the Amanda w ebsite.
2. **Bacula** is designed for automatic backup on heterogeneous netw orks. Can be rather complicated to use and recommended (by its authors) only to experienced administrators. Generally available on Enterprise Linux systems through usual repositories. Complete information can be found at the Bacula w ebsite.
3. **Clonezilla** is a very robust disk cloning program, w hich can make images of disks and deploy them, either to restore a backup, or to be used for ghosting, to provide an image that can be used to install many machines. Complete information can be found at the Clonezilla w ebsite.
   Program comes in tw o versions: Clonezilla live, w hich is good for single machine backup and recovery, and Clonezilla SE, server **e**dition, w hich can clone to many computers at the same time. Clonezilla not very hard to use, and extremely flexible, supporting many operating systems (not just Linux), filesystem types, and boot loaders.

##

Back to top

Previous Chapter - Table of Contents - Next Chapter