Backing Up and Compressing Data

Backing Up Data:

1. There are many ways you can back up data or even your entire system.
2. Basic ways to do so include the use of simple copying with cp and use of the more robust **rsync**. Both can be used to synchronize entire directory trees.
3. However, **rsync** is more efficient, because it checks if the file being copied already exists.
4. If the file exists and there is no change in size or modification time, **rsync** will avoid an unnecessary copy and save time.
5. Furthermore, because **rsync** copies only the parts of files that have actually changed, it can be very fast.
6. **cp** can only copy files to and from destinations on the local machine (unless you are copying to or from a filesystem mounted using **NFS**), but **rsync** can also be used to copy files from one machine to another.
7. Locations are designated in the **target:path** form, where target can be in the form of **someone@host**. The **someone@ part** is optional and used if the remote user is different from the local user.
8. **rsync** is very efficient when recursively copying one directory tree to another, because only the differences are transmitted over the network.
9. One often synchronizes the destination directory tree with the origin, using the **-r** option to recursively walk down the directory tree copying all files and directories below the one listed as the source.

Using rsync:

1. **rsync** is a very powerful utility.
2. For example, a very useful way to back up a project directory might be to use the following command:
   rsync -r project-X archive-machine:archives/project-X
3. Note that **rsync** can be very destructive! Accidental misuse can do a lot of harm to data and programs, by inadvertently copying changes to where they are not wanted.
4. Take care to specify the correct options and paths. It is highly recommended that you first test your **rsync** command using the **-dry-run** option to ensure that it provides the results that you want.
5. To use **rsync** at the command prompt, type **rsync sourcefile destinationfile**, where either file can be on the local machine or on a networked machine
6. The contents of **sourcefile** will be copied to **destinationfile**.
   rsync --progress -avrxH --delete sourcedir destdir

Compressing Data:

1. File data is often compressed to save disk space and reduce the time it takes to transmit files over networks.
2. Linux uses a number of methods to perform this compression, including:

Command	Usage
`gzip`	The most frequently used Linux compression utility
`bzip2`	Produces files significantly smaller than those produced by `gzip`
`xz`	The most space-efficient compression utility used in Linux
`zip`	Is often required to examine and decompress archives from other operating systems

3. These techniques vary in the efficiency of the compression (how much space is saved) and in how long they take to compress; generally, the more efficient techniques take longer.
4. Decompression time does not vary as much across different methods.
5. In addition, the tar utility is often used to group files in an archive and then compress the whole archive at once.

Compressing Data Using gzip:

1. **gzip** is the most often used Linux compression utility. It compresses very well and is very fast.
2. The following table provides some usage examples:

Command	Usage
`gzip *`	Compresses all files in the current directory; each file is compressed and renamed with a `.gz` extension.
`gzip -r projectX`	Compresses all files in the `projectX` directory, along with all files in all of the directories under `projectX`.
`gunzip foo`	De-compresses `foo` found in the file `foo.gz`. Under the hood, the `gunzip` command is actually the same as `gzip –d`.

Compressing Data Using bzip2:

1. bzip2 has a syntax that is similar to gzip but it uses a different compression algorithm and produces significantly smaller files, at the price of taking a longer time to do its work.
2. Thus, it is more likely to be used to compress larger files.
3. **NOTE:** bzip2 has lately become deprecated due to lack of maintenance and the superior compression ratios of xz which is actively maintained.
4. Examples of common usage are also similar to gzip:

Command	Usage
`bzip2 *`	Compresses all of the files in the current directory and replaces each file with a file renamed with a `.bz2` extension.
`bunzip2 *.bz2`	Decompresses all of the files with an extension of `.bz2` in the current directory. Under the hood, `bunzip2` is the same as calling `bzip2 -d`.

Compressing Data Using xz:

1. **xz** is the most space efficient compression utility used in Linux and is used to store archives of the Linux kernel.
2. Once again, it trades a slower compression speed for an even higher compression ratio.
3. Compressed files are stored with a **.xz** extension.

Command	Usage
`xz *`	Compresses all of the files in the current directory and replaces each file with one with a `.xz` extension.
`xz foo`	Compresses `foo` into `foo.xz` using the default compression level (-6), and removes `foo` if compression succeeds.
`xz -dk bar.xz`	Decompresses `bar.xz` into `bar` and does not remove `bar.xz` even if decompression is successful.
`xz -dcf a.txt b.txt.xz > abcd.txt`	Decompresses a mix of compressed and uncompressed files to standard output, using a single command.
`xz -d *.xz`	Decompresses the files compressed using `xz`.

Handling Files Using zip:

1. The **zip** program is not often used to compress files in Linux, but is often required to examine and decompress archives from other operating systems.
2. It is only used in Linux when you get a zipped file from a Windows user. It is a legacy program.

Command	Usage
`zip backup *`	Compresses all files in the current directory and places them in the `backup.zip`.
`zip -r backup.zip ~`	Archives your login directory (~) and all files and directories under it in `backup.zip`.
`unzip backup.zip`	Extracts all files in `backup.zip` and places them in the current directory.

Archiving and Compressing Data Using tar:

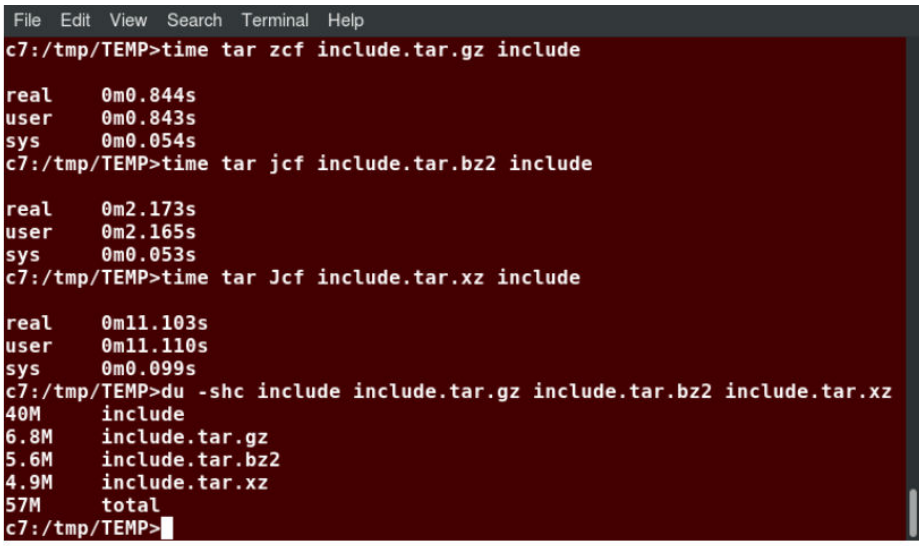
1. Historically, tar stood for "**tape archive**" and was used to archive files to a magnetic tape.
2. It allows you to create or extract files from an archive file, often called a **tarball**.

3. At the same time, you can optionally compress while creating the archive, and decompress while extracting its contents.
4. You can separate out the archiving and compression stages, as below but this is slower and wastes space by creating an unneeded intermediary **.tar** file.
    - **Ex:** tar cvf mydir.tar mydir ;
      gzip mydir.tar
    - **Ex:** gunzip mydir.tar.gz ;
      tar xvf mydir.tar

Command	Usage
`tar xvf mydir.tar`	Extract all the files in `mydir.tar` into the `mydir` directory.
`tar zcvf mydir.tar.gz mydir`	Create the archive and compress with `gzip`.
`tar jcvf mydir.tar.bz2 mydir`	Create the archive and compress with `bz2`.
`tar Jcvf mydir.tar.xz mydir`	Create the archive and compress with `xz`.
`tar xvf mydir.tar.gz`	Extract all the files in `mydir.tar.gz` into the `mydir` directory. ***NOTE**: You do **not** have to tell **tar** it is in `gzip` format.*

Relative Compression Times and Sizes:

1. To demonstrate the relative efficiency of gzip, bzip2, and xz, the following screenshot shows the results of compressing a purely text file directory tree (the include directory from the kernel source) using the three methods.
2. The below shows that as compression factors go up, CPU time does as well (i.e. producing smaller archives takes longer).



```
c7:/tmp/TEMP>time tar zcf include.tar.gz include

real    0m0.844s
user    0m0.843s
sys     0m0.054s
c7:/tmp/TEMP>time tar jcf include.tar.bz2 include

real    0m2.173s
user    0m2.165s
sys     0m0.053s
c7:/tmp/TEMP>time tar Jcf include.tar.xz include

real    0m11.103s
user    0m11.110s
sys     0m0.099s
c7:/tmp/TEMP>du -shc include include.tar.gz include.tar.bz2 include.tar.xz
40M     include
6.8M    include.tar.gz
5.6M    include.tar.bz2
4.9M    include.tar.xz
57M     total
c7:/tmp/TEMP>
```

Relative Compression Times and Sizes

Disk-to-Disk Copying (dd):

1. The **dd** program is very useful for making copies of raw disk space.
2. For example, to back up your Master Boot Record (MBR) (the first 512-byte sector on the disk that contains a table describing the partitions on that disk), you might type:
   ```
   dd if=/dev/sda of=sda.mbr bs=512 count=1
   ```
3. WARNING: Typing the below command to make a copy of one disk onto another, will delete everything that previously existed on the second disk.An exact copy of the first disk device is created on the second disk device.
4. **Note:** Do not experiment with this command as written above, as it can erase a hard disk!