Backups

Note that this is a very early draft and may contain erroneous data or unfinished sections. Comments, issues and tips are welcomed.

Backup are essential when working with all kinds of devices. While as a developer you work with tools like git that allow you to store code in the cloud, these services should under no circumstance be considered backups.

Archiving

GNU tar is an archiving program designed to store multiple files in a single file (an archive), and to manipulate such archives. The archive can be either a regular file or a device (e.g. a tape drive, hence the name of the program, which stands for tape archiver), which can be located either on the local or on a remote machine.

To create a basic tar archive (also called "tarballs"), you will need to the following syntax:

```
tar -cvf backup.tar <dir|files>
```

- `-c`: create tar archive
- `-v`: verbose (show what files are being handled)
- `-f <filename.tar>`: filename to use
- `<dir|files>`: directory or files to put into the archive

Now to check what files you archived, you can use the `--list` option:

```
tar --list -f backup.tar
```

Extracting

Extracting an archive is the reverse processed of archiving. It unpacks the tarball and copies the content back to the filesystem.

It's syntax is the same as the archiving syntax, except that it replaces the `-c` (create) flag with an `-x` (extract) flag.

```
tar -xvf backup.tar
```

Compressing Archive Files

Notice that the tar file is about the same size as the files it archives. This is because tar only archives the file and does not apply any compression by default.

Different options exists when compressing tar files.

gzip

gzip is based on the DEFLATE algorithm, which is a combination of LZ77 and Huffman coding. DEFLATE was intended as a replacement for LZW and other patent-encumbered data compression algorithms which, at the time, limited the usability of compress and other popular archivers.

Although its file format also allows for multiple such streams to be concatenated (gzipped files are simply decompressed concatenated as if they were originally one file), gzip is normally used to compress just single files. Compressed archives are typically created by assembling collections of files into a single tar archive (also called tarball), and then compressing that archive with gzip. The final compressed file usually has the extension `.tar.gz` or `.tgz`.

Various implementations of the program have been written. The most commonly known is the GNU Project's implementation using Lempel-Ziv coding (LZ77).

The amount of compression obtained depends on the size of the input and the distribution of common substrings. Typically, text such as source code or English is reduced by 60-70%. Compression is generally much better than that achieved by LZW (as used in `compress`), Huffman coding (as used in `pack`), or adaptive Huffman coding (`compact`).

The gzip format is used in HTTP compression, a technique used to speed up the sending of HTML and other content on the World Wide Web. It is one of the three standard formats for HTTP compression as specified in RFC 2616. This RFC also specifies a zlib format (called "DEFLATE"), which is equal to the gzip format except that gzip adds eleven bytes of overhead in the form of headers and trailers. Still, the gzip format is sometimes recommended over zlib because Internet Explorer does not implement the standard correctly and cannot handle the zlib format as specified in RFC 1950.

To create a tarball and compress using `gzip`, add the `-z` flag to the options. Also make sure to add the `.gz` file extension after `tar`.

```
tar -czvf backup.tar.gz <dir|files>
```

And again to extract, just replace the `-c` flag with the `-x` flag.

```
tar -xzvf backup.tar.gz
```

bzip2

bzip2 is a free and open-source file compression program that uses the Burrows–Wheeler algorithm. It only compresses single files and is not a file archiver. It was developed by Julian Seward, and maintained by Mark Wielaard and Micah Snyder.

bzip2 compresses files using the Burrows-Wheeler block sorting text compression algorithm, and Huffman coding. Compression is generally considerably better than that achieved by more conventional LZ77/LZ78-

based compressors, and approaches the performance of the PPM family of statistical compressors.

Since the late 1990s, bzip2, a file compression utility based on a block-sorting algorithm, has gained some popularity as a gzip replacement. It produces considerably smaller files (especially for source code and other structured text), but at the cost of memory and processing time (up to a factor of 4).

The compressed blocks in bzip2 can be independently decompressed, without having to process earlier blocks. This means that bzip2 files can be decompressed in parallel, making it a good format for use in big data applications with cluster computing frameworks like Hadoop and Apache Spark.

bzip2 compresses most files more effectively than the older LZW (.Z) and Deflate (.zip and .gz) compression algorithms, but is considerably slower.

bzip2 performance is asymmetric, as decompression is relatively fast. Motivated by the long time required for compression, a modified version was created in 2003 called pbzip2 that used multi-threading to encode the file in multiple chunks, giving almost linear speedup on multi-CPU and multi-core computers.

Like gzip, bzip2 is only a data compressor. It is not an archiver like tar or ZIP; the program itself has no facilities for multiple files, encryption or archive-splitting, but, in the UNIX tradition, relies instead on separate external utilities such as tar and GnuPG for these tasks.

To create a tarball and compress using `bzip2`, add the `-j` flag to the options. Also make sure to add the `.bz2` file extension after `tar`.

```
tar -cjvf backup.tar.bz2 <dir|files>
```

And again to extract, just replace the `-c` flag with the `-x` flag.

```
tar -xjvf backup.tar.bz2
```

What compression to use?

Source: https://superuser.com/questions/205223/pros-and-cons-of-bzip-vs-gzip

Gzip and bzip2, as well as xz and lzop, are functionally equivalent. (There once was a bzip, but it seems to have completely vanished off the face of the world.) Other common compression formats are zip, rar and 7z; these three do both compression and archiving (packing multiple files into one). Here are some typical ratings in terms of speed, availability and typical compression ratio (note that these ratings are somewhat subjective, don't take them as gospel):

```
decompression speed (fast > slow): lzop > gzip, zip > xz > 7z > rar > bzip2
compression speed (fast > slow): lzop > gzip, zip > xz > bzip2 > 7z > rar
compression ratio (better > worse): xz > 7z > rar, bzip2 > gzip > zip >
lzop
availability (unix): gzip > bzip2 > xz > lzop > zip > 7z > rar
availability (windows): zip > rar > 7z > gzip > bzip2, lzop, xz
```

Excluding files and directories

You can easily exclude some files or directories by specifying them using the `--exclude` flag. It takes a `glob(3)` style pattern.

```
tar --exclude='node_modules' -cvf backup.tar source
```

If you wish to keep the parent directory `node_modules` but wish to exclude all underlying files/directories you can also use:

```
tar --exclude='node_modules/*'-cvf backup.tar source
```

Beware that `--exclude=dir/ignore_this_dir` will match in any subtree as well! You'll end up missing files you didn't expect to be excluded.

If you wish to exclude specific directories at a given level, you will need to specify them using a relative path as shown below:

```
tar --exclude='source/course_linux_essentials/node_modules' -cvf backup.tar
source
```

Note that single quotes are added around the exclude pattern to make sure the shell is not globbing it.

You can also exclude multiple dirs/files:

```
tar --exclude='node_modules' --exclude='.git' -zcvf linux_course.tar.gz
course_linux_essentials
tar --exclude='node_modules' --exclude='.git' -zcvf projects_backup.tar.gz
projects
```

Exclude files

One can also use an exclude file, similar to a `.gitignore` file, to exclude multiple files and patterns.

```
tar -X ".exclude" -cvf backup.tar source
```

Where the file `.exclude` could contain the following content:

```
.vscode
.git
```

```
node_modules
mbed-os
```

Secure Copy

Secure Copy, or `scp`, allows files to be copied to, from, or between different hosts. It uses ssh for data transfer and provides the same authentication and same level of security as ssh.

The syntax of the `scp` command is as follows:

```
scp [[user1@]host1:]<source> [[user2@]host2:]<destination>
```

Remote Syncing

`rsync` is a utility for efficiently transferring and synchronizing files between a computer and a storage drive and across networked computers by comparing the modification times and sizes of files. It is commonly found on Unix-like operating systems and is under the GPL-3.0-or-later license.

It can copy locally, to/from another host over any remote shell, or to/from a remote rsync daemon. It offers a large number of options that control every aspect of its behavior and permit very flexible specification of the set of files to be copied. It is famous for its delta-transfer algorithm, which reduces the amount of data sent over the network by sending only the differences between the source files and the existing files in the destination. Rsync is widely used for backups and mirroring and as an improved copy command for everyday use.

Syntax:

```
rsync [OPTION...] SRC... [DEST]
```

Some options

- `-r`: recurse into directories
- `-a`: archive mode ()
- `-n`: dry-run, test it out first
- `-v`: verbose, more information
- `-z`: use compression
- `-P`: show progress bar and allow resume
- `--delete`: remove files in destination if not present in source (truly sync)

Useful option is to use the `-a` flag, which is a combination flag and stands for “archive”. This flag syncs recursively and preserves symbolic links, special and device files, modification times, groups, owners, and permissions. It’s more commonly used than `-r` and is the recommended flag to use.

Examples:

```
rsync -anv source/ destination
rsync -a source/ destination
rsync -aPv source/ destination
```

Notice the trailing slash after the source directory. This is required if you wish to sync both dirs directly. Otherwise a `source` directory would be created below `destination`.

Rsync also support excluding files/directories using `--exclude=PATTERN` or using an exclude file specified with the flag `--exclude-from=FILE`.