- Base64 is a group of similar encoding schemes that represents binary data in an ASCII string format by translating it into a radix-64 representation. The `base64` command can be used to encode and decode the Base64 string. In order to encode a binary file into the Base64 format, use:

  ```
  $ base64 filename > outputfile
  ```

  Or:

  ```
  $ cat file | base64 > outputfile
  ```

  It can read from `stdin`.

  Decode Base64 data as follows:

  ```
  $ base64 -d file > outputfile
  ```

  Or:

  ```
  $ cat base64_file | base64 -d > outputfile
  ```

- **md5sum** and **SHA-1** are unidirectional hash algorithms, which cannot be reversed to form the original data. These are usually used to verify the integrity of data or for generating a unique key from a given data:

  ```
  $ md5sum file
  8503063d5488c3080d4800ff50850dc9  file


  $ sha1sum file
  1ba02b66e2e557fede8f61b7df282cd0a27b816b  file
  ```

  These types of hashes are commonly used for storing passwords. Passwords are stored as their hashes and when a user wants to authenticate, the password is read and converted to the hash. Then, this hash is compared to the one that is stored already. If they are the same, the password is authenticated and access is provided, otherwise it is denied. Storing plain text password strings is risky and poses a security risk.

Although commonly used, md5sum and SHA-1 are no longer considered secure. This is because of the rise of computing power in recent times that makes it easier to crack them. It is recommended to use tools such as `bcrypt` or `sha512sum` instead. Read more about this at `http://codahale.com/how-to-safely-store-a-password/`.

- **Shadow-like hash (salted hash)**

  Let us see how to generate a shadow-like salted hash for passwords. The user passwords in Linux are stored as their hashes in the `/etc/shadow` file. A typical line in `/etc/shadow` will look like this:

  ```
  test:$6$fG4eWdUi$ohTKOlEUzNk77.4S8MrYe07NTRV4M3LrJnZP9p.qc1bR5c.
  EcOruzPXfEu1uloBFUa18ENRH7F70zhodas3cR.:14790:0:99999:7:::
  ```

  `$6$fG4eWdUi$ohTKOlEUzNk77.4S8MrYe07NTRV4M3LrJnZP9p.qc1bR5c.EcOruzPXfEu1uloBFUa18ENRH7F70zhodas3cR` is the shadow hash corresponding to its password.

  In some situations, we may need to write critical administration scripts that may need to edit passwords or add users manually using a shell script. In that case we have to generate a shadow password string and write a similar line as the preceding one to the shadow file. Let's see how to generate a shadow password using `openssl`.

  Shadow passwords are usually salted passwords. `SALT` is an extra string used to obfuscate and make the encryption stronger. The salt consists of random bits that are used as one of the inputs to a key derivation function that generates the salted hash for the password.

For more details on salt, see the Wikipedia page `http://en.wikipedia.org/wiki/Salt_(cryptography)`.

```
$ opensslpasswd -1 -salt SALT_STRING PASSWORD
$1$SALT_STRING$323VkWkSLHuhbt1zkSsUG.
```

Replace `SALT_STRING` with a random string and `PASSWORD` with the password you want to use.

Sorting unique and duplicates

Sorting is a common task that we can encounter with text files. The `sort` command helps us to perform sort operations over text files and `stdin`. Most often, it can also be coupled with many other commands to produce the required output. `uniq` is another command that is often used along with a `sort` command. It helps to extract unique (or duplicate) lines from a text or `stdin`. This recipe illustrates most of the use cases with `sort` and `uniq` commands.