An example of the usage is as follows:

```
fpath="/etc/passwd"
if [ -e $fpath ]; then
    echo File exists;
else
    echo Does not exist;
fi
```

- String comparisons: While using string comparison, it is best to use double square brackets, since the use of single brackets can sometimes lead to errors.

  Two strings can be compared to check whether they are the same in the following manner:

  - `[[ $str1 = $str2 ]]`: This returns true when `str1` equals `str2`, that is, the text contents of `str1` and `str2` are the same
  - `[[ $str1 == $str2 ]]`: It is an alternative method for string equality check

  We can check whether two strings are not the same as follows:

  - `[[ $str1 != $str2 ]]`: This returns true when `str1` and `str2` mismatch

  We can find out the alphabetically smaller or larger string as follows:

  - `[[ $str1 > $str2 ]]`: This returns true when `str1` is alphabetically greater than `str2`
  - `[[ $str1 < $str2 ]]`: This returns true when `str1` is alphabetically lesser than `str2`

  > Note that a space is provided after and before `=`, if it is not provided, it is not a comparison, but it becomes an assignment statement.

  - `[[ -z $str1 ]]`: This returns true if `str1` holds an empty string
  - `[[ -n $str1 ]]`: This returns true if `str1` holds a nonempty string

  It is easier to combine multiple conditions using logical operators such as `&&` and `||` in the following code:

```
if [[ -n $str1 ]] && [[ -z $str2 ]] ;
then
    commands;
fi
```

For example:

```
str1="Not empty "
str2=""
if [[ -n $str1 ]] && [[ -z $str2 ]];
then
    echo str1 is nonempty and str2 is empty string.
fi
```

Output:

```
str1 is nonempty and str2 is empty string.
```

The test command can be used for performing condition checks. It helps to avoid usage of many braces. The same set of test conditions enclosed within `[]` can be used for the test command.

For example:

```
if  [ $var -eq 0 ]; then echo "True"; fi
can be written as
if  test $var -eq 0 ; then echo "True"; fi
```