IFOPT README

`ifopt` is my own **Command Line Interface** options parser with some other feature. During a project where I developed a simple script, I need to add some option to change the behavior. I use **NodeJS** options and parse them as like **PHP** perform.

So, finaly I created `ifopt` to reuse it event if **cli** lib already exist.

Summary

How to use it

Installation of `ifopt`

- Go into the root of your project
- Type the following command : `npm install ifopt` :
    - That will creates a `package.json` file, or add it as dependency.

Load `ifopt` object

- Once `ifopt` installed, load it as following :

```
const opt = require('ifopt');
```

`ifopt` mains functionnalities

In the world of **Command Line Interface options**, there is two kind of options :

- The short options which begins with one dash (`-`).
- The long options which begins with two dashes (`--`).

I created a third kind of option : **the implicits** ones. All elements put behind the command are an option.

For instance, in this command `find text -v --output=file.txt` , `text` is also an option as `-v` and `--output` are. **implicits** option are identified with their position, without taking account of short & long option :

- `find text -v --output=file.txt`
- `find -v text --output=file.txt`
- `find -v --output=file.txt text`

Herebefore, `text` is always the **first** implicit option.

`ifopt` only parse options. Using returned option is in your hand. You can decide to use implicit, sort and long option for the same information (Eg: **input file**) .

An option **CAN HAVE** (`::`), **MUST HAVE** (`:`) or **NOT** ( ) a value. It's possible to set the expected behavior regarding the option. `ifopt` will warn the user when the option not fullfill the requirement.

Parse NodeJS options

`ifopt` offers differents ways to set and get **NodeJS** options. The simpliest way is to get options is parse them directly using your options configuration :

```
const opt = require('ifopt');

// Declaration of option which will be manage by ifopt :
const myOptions = {
    shortopt: "hd:o::",
    longopt: [
        "help",     // short is h (NOT  HAVE a value)
        "dir:" ,    // short is d (MUST HAVE a value)
        "output::"  // short is o (CAN  HAVE a value)
    ]
}

let parsedOption = opt.getopt(
    myOptions.shortopt,
    myOptions.longopt
);
```

The following execution with this command will return for `parsedOption` :

```
myCommand -d=test -o --unwantedoption
```

```
{
  d: { arg: '-d=test', opt: 'd', val: 'test' },
  o: { arg: '-o', opt: 'o', val: null }
}
```

You can separately configure `ifopt` :

```
const opt = require('ifopt');

// Set Short Options
opt.setShortOpt("hd:o::");
// Set Long Options
opt.setLongOpt([
    "help",     // short is h (NOT  HAVE a value)
    "dir:" ,    // short is d (MUST HAVE a value)
    "output::"  // short is o (CAN  HAVE a value)
]);

let parsedOption = opt.getopt();
```

Which will produce the same result :

```
{
  d: { arg: '-d=test', opt: 'd', val: 'test' },
  o: { arg: '-o', opt: 'o', val: null }
}
```

Another way is to use `setOpts` :

```
const opt = require('ifopt');

// Set Short Options
opt.setOpts(
    // Short Ones
    "hd:o::",
    // Long Ones
    [
        "help",      // short is h (NOT  HAVE a value)
        "dir:" ,     // short is d (MUST HAVE a value)
        "output::"  // short is o (CAN  HAVE a value)
    ]
);

let parsedOption = opt.getopt();
```

Focus on implicits

Please find below how to handle implicits options for the following command :

```
myCommand myInputFile --dir=test myOutputFile
```

```
const opt = require('ifopt');

// Declaration of option which will be manage by ifopt :
const myOptions = {
    shortopt: "hd:o::i:",
    longopt: [
        "help",      // short is h (NOT  HAVE a value)
        "dir:" ,     // short is d (MUST HAVE a value)
        "input:" ,   // short is d (MUST HAVE a value)
        "output::"  // short is o (CAN  HAVE a value)
    ]
}

let implicitsHandler = {
    implicitOneForInput : null,
    implicitTwoForOutput: null
}

// Parse command line arguments
let parsedOption = opt.getopt(
    myOptions.shortopt,                                     // Short Options
    myOptions.longopt,                                        // Long  Options
    ['implicitOneForInput', 'implicitTwoForOutput'],    // Implicits Order for Handler
    implicitsHandler                                        // Implicits Handler to get Data by Ref
);

// Result of command : myCommand myInputFile --dir=test myOutputFile
console.log(parsedOption);
console.log(implicitsHandler);
```

will return :

```
{ dir: { arg: '--dir=test', opt: 'dir', val: 'test' } }
{
  implicitOneForInput: 'myInputFile',
  implicitTwoForOutput: 'myOutputFile'
}
```

So you can also pass implicit order and handler for method `setOpts` :

```
let implicitsHandler = {
    implicitOneForInput : null,
    implicitTwoForOutput: null
}

opt.setOpts(
    // Short Ones
    "hd:o::",
    // Long Ones
    [
        "help",      // short is h (NOT  HAVE a value)
        "dir:" ,     // short is d (MUST HAVE a value)
        "output::"  // short is o (CAN  HAVE a value)
    ],
    // Implicits Orders
    ['implicitOneForInput', 'implicitTwoForOutput'],
    // Implicits Handler
    implicitsHandler
);
```

You can also set implicits separately :

```
let implicitsHandler = {
    implicitOneForInput : null,
    implicitTwoForOutput: null
}

// Set Implicits
opt.setImplicitOpt(
    ['implicitOneForInput', 'implicitTwoForOutput'],
    implicitsHandler
)
```

Check Option (isOption())

Once you have parsed arguments from your command line, you have to developped your logique and your function to perform processing.

Sometime, you want to check if an option have been correctly passed.

Instead of checking in `parsedOptions` get from `getopt`, you can directly use method `isOption()` which can check one or more options at once. Done like this, you are checking for one option which can be provided in long or short version :

```
if (opt.isOption(['dir','d'])) {
    console.log("Directory is provided")
} else {
    console.log("Directory is NOT provided")
}
```

If you want to combine availability of two or more options you can change the operator :

```
if (opt.isOption(['d','i'], 'and')
) {
    console.log("Directory AND input are provided")
} else {
    console.log("Directory OR input NOT provided")
}
```

Get Option Value (getOptValue())

Once again, to prevent you to get value in `parsedOptions`, you can use method `getOptValue` :

```
let directory = opt.getOptValue(['dir', 'd']);
```

The herebefore statement will return the value of the first option found. It's very usefull to get value independantly of the short and long option. Done like this, longs options have the priority over shorts ones.

For command :

```
myCommand -d=test --dir=test2
```

`directory` will be equal to `test2` (`--dir=test2`)

Get Option ValueS (getOptsValues())

This version will return an array of values for provided options :

```
let files = opt.getOptsValues(['input', 'i']);
```

For command :

```
myCommand -i=file_1 --input=file_2 -i=file_3
```

`files` is equal to `[ 'file_2', 'file_1', 'file_3' ]`

Log in STDOUT (log())

`ifopt` provides a method to send message in `STDOUT` nammed `log()` .

By default, this command will generate a message like this in the console using colors :

```
[ level ]: message
```

Below, the argument of method `log` :

- String, message, Message to display.
- Number, level, Level of the message. 0=OK,1=KO,2=WARN.
- Array, args Arguments which will replace placeholder (%s) in message.

```
const log = opt.log;
// Considering provided option "i" was equal to "<yourFile>"
log("File %s not found", 1, [opt.getOptValue("i")]);
```

The herebefore statement will log the message :

```
[ERROR]: File <yourFile> not found
```

Enabling / Disabling Colors

When you redirect you output to file, you do not want control char in your log file.

So, to prevent specials char, you can easily disable colors.

Simply call method `noColor()` to turn off color in method `log()` .

```
opt.noColor();
// Or like this
opt.useColor(false);
```

To turn on the color :

```
opt.useColor();
// Or
opt.useColor(true);
```

Managing colors

Get default colors (getColors())

Get one color (getColor())

Update colors (setColors())

Update one color (setColor())