CLI

Files in this folder define the common framework for defining and parsing CLI commands, options and properties such as `inApp` or `inPackage`.

It is encouraged to split out theme sharing commands into separate files.

An example of a command:

```
main.registerCommand({
  name: 'some-command',
  requiresRelease: false,
  requiresApp: true,
  // requiresPackage: true,
  // requiresAppOrPackage: true,
  pretty: true, // optional
  minArgs: 1, // optional
  maxArgs: 10, // optional
  options: {
    'long-option': { type: Boolean, short: 's', default: true, required: true },
    'with-string': { type: String, short: 'w' }
  }
}, function (options) {
  var {appDir, packageDir, args} = options;
  var longOption = options['long-option'];
  var withString = options['with-string'];
  ...
});
```

This command will handle the following examples:

```
meteor some-command --long-option
meteor some-command -s
meteor some-command -s --with-string "some value"
meteor some-command -s -w "some value"
```

Note: don't pick a short key for an option unless it is very common to use it. The commands parser makes sure the same short key for an option is used consistently for the same kind of option across all commands. So two commands A and B can share a short option "c" only if it stands for the same long option in both cases.

Catalog refresh policy

You might notice that some commands are marked to never refresh the catalog. This is useful to make the command fast, if the command doesn't involve any network communication to the package server, it might be a good idea to forbid the preemtive refresh:

```
main.registerCommand({
  ...
  catalogRefresh: new catalog.Refresh.Never()
});
```

Command return code

For some reason there are two ways to terminate the command with a non-standard exit code (non-zero, unix process exit code).

First way is just the return value of the command function, which should be a number.

The second way is to throw a special kind of exception:

```
main.registerCommand({...}, function (options) {
  throw new main.ExitWithCode(2);
})
```

In case of additional options check, you can throw another special exception to show the help text for the command:

```
main.registerCommand({...}, function (options) {
  if (options.bla === 'bad-value')
    throw new main.ShowUsage;
})
```

There is also `main.WaitForExit` used for terminating after the subprocesses are done. (XXX more info)

help.txt

The `help.txt` file contains all the information printed to the user when the `meteor help` command is run. It has its own syntax where every command is separated by a marker `>>>` with the name of the command.

Commands are parsed so the help text for an individual command could be separated from the rest.

Sadly, these commands need to be redocumented in the docs app as well. The docs text is not shared between these two places, yet.

Admin commands

Given that `self-test` is the only reliable way to test code in Meteor Tool, you might need to create special "admin" commands. Since "admin" commands don't show up in the help text, this namespace is abused to store all sorts of one-of commands to run in automated and manual tests.