Async Completion

Node libraries handle asynchronicity in a variety of ways. The most common pattern is error-first callbacks, but you might also encounter streams, promises, event emitters, child processes, or observables. Gulp tasks normalize all these types of asynchronicity.

Signal task completion

When a stream, promise, event emitter, child process, or observable is returned from a task, the success or error informs gulp whether to continue or end. If a task errors, gulp will end immediately and show that error.

When composing tasks with series(), an error will end the composition and no further tasks will be executed. When composing tasks with parallel(), an error will end the composition but the other parallel tasks may or may not complete.

Returning a stream

```
const { src, dest } = require('gulp');

function streamTask() {
  return src('*.js')
    .pipe(dest('output'));
}

exports.default = streamTask;
```

Returning a promise

```
function promiseTask() {
  return Promise.resolve('the value is ignored');
}

exports.default = promiseTask;
```

Returning an event emitter

```
const { EventEmitter } = require('events');
```

```
function eventEmitterTask() {
  const emitter = new EventEmitter();
  // Emit has to happen async otherwise gulp isn't listening yet
  setTimeout(() => emitter.emit('finish'), 250);
  return emitter;
}

exports.default = eventEmitterTask;
```

Returning a child process

```
const { exec } = require('child_process');

function childProcessTask() {
  return exec('date');
}

exports.default = childProcessTask;
```

Returning an observable

```
const { Observable } = require('rxjs');

function observableTask() {
  return Observable.of(1, 2, 3);
}

exports.default = observableTask;
```

Using an error-first callback

If nothing is returned from your task, you must use the error-first callback to signal completion. The callback will be passed to your task as the only argument - named cb() in the examples below.

```
function callbackTask(cb) {
  // `cb()` should be called by some async work
  cb();
}

exports.default = callbackTask;
```

To indicate to gulp that an error occurred in a task using an error-first callback, call it with an Error as the only argument.

```
function callbackError(cb) {
  // `cb()` should be called by some async work
  cb(new Error('kaboom'));
}

exports.default = callbackError;
```

However, you'll often pass this callback to another API instead of calling it yourself.

```
const fs = require('fs');

function passingCallback(cb) {
  fs.access('gulpfile.js', cb);
}

exports.default = passingCallback;
```

No synchronous tasks

Synchronous tasks are no longer supported. They often led to subtle mistakes that were hard to debug, like forgetting to return your streams from a task.

When you see the *"Did you forget to signal async completion?"* warning, none of the techniques mentioned above were used. You'll need to use the error-first callback or return a stream,

promise, event emitter, child process, or observable to resolve the issue.

Using async/await

When not using any of the previous options, you can define your task as an async function, which wraps your task in a promise. This allows you to work with promises synchronously using await and use other synchronous code.

```
const fs = require('fs');

async function asyncAwaitTask() {
  const { version } = JSON.parse(fs.readFileSync('package.json', 'u
  console.log(version);
  await Promise.resolve('some result');
}

exports.default = asyncAwaitTask;
```