gatsby-core-utils

Utilities used in multiple Gatsby packages.

Usage

```
npm install gatsby-core-utils
```

createContentDigest

Encrypts an input using md5 hash of hexadecimal digest.

```
const { createContentDigest } = require("gatsby-core-utils")

const options = {
  key: "value",
  foo: "bar",
}

const digest = createContentDigest(options)
// ...
```

cpuCoreCount

Calculate the number of CPU cores on the current machine

This function can be controlled by an env variable `GATSBY_CPU_COUNT` setting the first argument to true.

value	description
	Counts amount of real cores by running a shell command
logical-cores	`require("os").cpus()` to count all virtual cores
any number	Sets cpu count to that specific number

```
const { cpuCoreCount } = require("gatsby-core-utils")

const coreCount = cpuCoreCount(false)
// ...
```

```
const { cpuCoreCount } = require("gatsby-core-utils")
process.env.GATSBY_CPU_COUNT = "logical-cores"

const coreCount = cpuCoreCount()
// ...
```

joinPath

A utility that joins paths with a `/` on windows and unix-type platforms. This can also be used for URL concatenation.

```
const { joinPath } = require("gatsby-core-utils")

const BASEPATH = "/mybase/"
const pathname = "./gatsby/is/awesome"
const url = joinPath(BASEPATH, pathname)
// ...
```

isCI

A utility that enhances `isCI` from 'ci-info' with support for Vercel and Heroku detection

```
const { isCI } = require("gatsby-core-utils")

if (isCI()) {
  // execute CI-specific code
}
// ...
```

getCIName

A utility that returns the name of the current CI environment if available, `null` otherwise

```
const { getCIName } = require("gatsby-core-utils")

const CI_NAME = getCIName()
console.log({ CI_NAME })
// {CI_NAME: null}, or
// {CI_NAME: "Vercel"}
// ...
```

createRequireFromPath

A cross-version polyfill for Node's `Module.createRequire`.

```
const { createRequireFromPath } = require("gatsby-core-utils")

const requireUtil = createRequireFromPath("../src/utils/")

// Require `../src/utils/some-tool`
requireUtil("./some-tool")
// ...
```

Mutex

When working inside workers or async operations you want some kind of concurrency control that a specific work load can only concurrent one at a time. This is what a Mutex does.

By implementing the following code, the code is only executed one at a time and the other threads/async workloads are awaited until the current one is done. This is handy when writing to the same file to disk.

```
const { createMutex } = require("gatsby-core-utils/mutex")

const mutex = createMutex("my-custom-mutex-key")
await mutex.acquire()

await fs.writeFile("pathToFile", "my custom content")

await mutex.release()
```