

# NODE JS

FICHIERS

# NODE JS

## Fichiers

- Manipuler des Chemins
- Manipuler des dossiers
- Manipuler des fichiers

Manipuler des chemins

# Manipuler des chemins

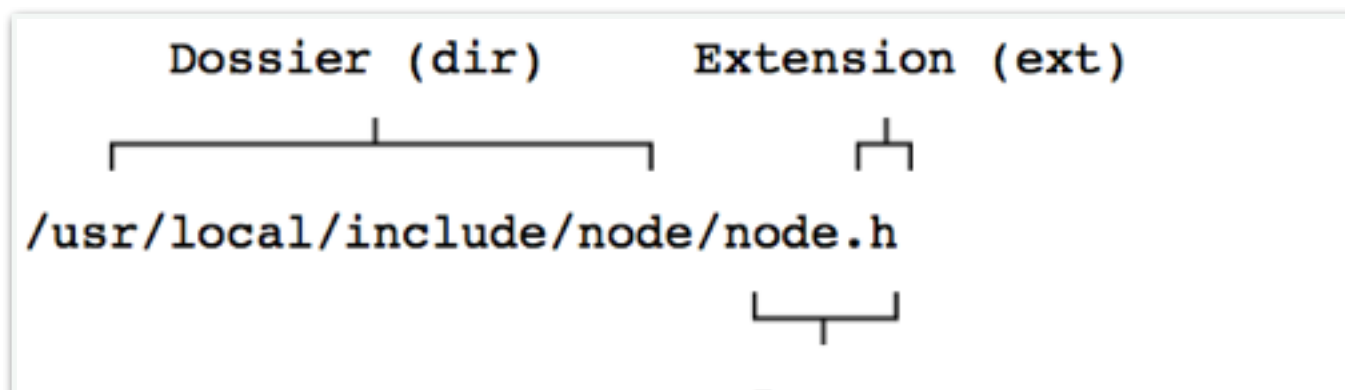
- Module Path :

***var pathLib = require('path');***

Contrairement à la pratique habituelle, qui veut que l'on nomme la variable symbolisant le module comme celui-ci, on utilise `pathLib` à la place de `path` car ce dernier est trop commun et risque d'entrer en conflit avec les variables locales.

# Manipuler des chemins

- **dirname**(path), **basename**(path, ext), **extname**(path)



The diagram illustrates the decomposition of the path `/usr/local/include/node/node.h` into its directory and extension components. A horizontal line with a vertical tick in the middle represents the path. The left side is labeled "Dossier (dir)" and the right side is labeled "Extension (ext)". Below the path, the full path is written: `/usr/local/include/node/node.h`. A bracket below the path indicates the "Base" component, which is the part of the path without the extension.

```
var path = '/usr/local/include/node/node.h';  
  
var dir = pathLib.dirname(path);  
// → /usr/local/include/node  
  
var base = pathLib.basename(path);  
// → node.h  
  
var ext = pathLib.extname(path);  
// → .h
```

# Manipuler des chemins

- `normalize(path)`

```
var path = pathLib.normalize('foo/../../bar/./baz//')  
  
console.log(path);  
//    →    ../../bar/baz/
```

La fonction `normalize(path)` nettoie un chemin en retirant les séparateurs en trop et en remplaçant les références aux dossiers courant (.) et parent (..) autant que possible.

Manipuler des dossiers

# Manipuler des dossiers

- Module fs

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

fs est le module contenant toutes les fonctionnalités permettant de manipuler des dossiers.



# Manipuler des dossiers

- Lecture d'un répertoire : ***readdir***

```
fs.readdir('./mondossier', function (error, files) {  
  if (error) {  
    console.error('échec de lecture du répertoire', error);  
  } else {  
    console.log('fichiers trouvés :', files);  
  }  
});
```

La lecture du contenu d'un répertoire se fait grâce à la fonction `readdir`, prenant comme paramètres le chemin du répertoire et une callback. Cette dernière affiche la liste des fichiers trouvés, ou une erreur en cas d'échec de la lecture du dossier.

# Manipuler des dossiers

- Création d'un répertoire : ***mkdir***

```
fs.mkdir('./logs', function (error) {  
  if (error) {  
    console.error('échec de création du répertoire', error);  
  } else {  
    console.log('répertoire créé');  
  }  
});
```

La création d'un seul répertoire est réalisée par la fonction `mkdir`, bien connue sous un système Unix. Elle demande deux paramètres, comme la fonction précédente : le chemin du dossier à créer, et une callback pour afficher le résultat.

# Manipuler des dossiers

- Créer une arborescence : module **mkdirp**
- <http://npmjs.org/package/mkdirp>

```
var mkdirp = require('mkdirp');

mkdirp('./project/logs/1969/09', function (error) {
  if (error) {
    console.error('échec de création de l\'arborescence', error);
  } else {
    console.log('arborescence créée');
  }
});
```

Ce module est calqué sur le même principe que les fonctions précédentes : en paramètres, une arborescence et une callback

# Manipuler des dossiers

- Supprimer un dossier : ***rmdir***

```
fs.rmdir('./logs', function (error) {  
  if (error) {  
    console.error('échec de suppression du répertoire', error);  
  } else {  
    console.log('répertoire créé');  
  }  
});
```

La suppression d'un répertoire vide se fait via la fonction `rmdir`. On passe le chemin du répertoire à supprimer et une callback, comme dans l'exemple ci-après.

# Manipuler des dossiers

- Supprimer un répertoire non vide et tout son contenu : module **rimraf**
- <http://npmjs.org/package/rimraf>

```
var rimraf = require('rimraf');

rimraf('./logs', function (error) {
  if (error) {
    console.error('échec de la suppression récursive du
répertoire', error);
  } else {
    console.log('répertoire récursivement supprimé');
  }
});
```

Manipuler des fichiers

# Manipuler des fichiers

- Module `fs` : toutes les fonctions nécessaires à la manipulation des fichiers

```
var file : require('fs');
```

Le paquet `graceful-fs` (<http://npmjs.org/package/graceful-fs>) propose la même API que le module `fs`, mais améliore la portabilité et est plus robuste aux erreurs.

# Manipuler des fichiers

- Tester si un fichier ou répertoire existe : **exists()**

```
fs.exists('/tmp/', function (doesExist) {  
  if (doesExist) {  
    console.log('le fichier existe');  
  } else {  
    console.log('le fichier n\'existe pas');  
  }  
});
```

La fonction `exists` permet de tester simplement l'existence d'un fichier (en réalité, toute entrée dans un système de fichiers : fichier, dossier, etc.). On lui passe le chemin vers le fichier à vérifier et une callback.

Pour des raisons historiques, contrairement à la convention dans Node, cette fonction ne passe pas d'erreur en premier argument à la fonction de rappel.



# Manipuler des fichiers

- Lire les métadonnées : **stat()**

```
fs.stat('/home/sandra', function (error, stats) {  
  if (error) {  
    console.error('échec de récupération des métadonnées', error);  
    return;  
  }  
  
  var type =  
    stats.isFile() ? 'fichier' :  
    stats.isDirectory() ? 'dossier' :  
    'inconnu'  
  ;  
  
  console.log('Ce fichier est de type %s.', type);  
  console.log('Il a une taille de %s octets.', stats.size);  
});
```

La fonction `stat(path, callback)` permet de récupérer les métadonnées d'un fichier. Les métadonnées sont disponibles dans une instance de la classe `Stat`.

# Manipuler des fichiers

- Modifier le propriétaire et groupe : **chown()**

```
fs.chown('/var/www', 0, 0, function (error) {  
  if (error) {  
    console.error('échec du changement de propriétaire', error);  
  } else {  
    console.log('propriétaire changé');  
  }  
});
```

La fonction `chown(path, uid, gid, callback)` permet de changer le propriétaire et le groupe d'un fichier. Au niveau des paramètres, il faut le chemin vers le fichier en premier argument, puis l'`uid` et le `gid` souhaités, enfin l'éternelle `callback`. Notez qu'il n'est pas possible d'utiliser une chaîne de caractères (nom du propriétaire ou du groupe) : on doit utiliser les identifiants numériques de l'utilisateur et du groupe souhaités.

# Manipuler des fichiers

- Modifier les permissions d'un fichier : **chmod()**

```
fs.chmod('/', 0666, function (error) {  
  if (error) {  
    console.error('échec du changement de mode', error);  
  } else {  
    console.log('mode changé');  
  }  
});
```

Pour changer le mode d'un fichier (les permissions), c'est très logiquement `chmod(path, mode, callback)` qui est utilisé, avec le passage habituel de paramètres. C'est le même mode existant sous Unix et ses dérivés.

# Manipuler des fichiers

- Modifier les dates d'accès et de modification : **utimes()**

```
fs.chmod('/', 0666, function (error) {  
  if (error) {  
    console.error('échec du changement de mode', error);  
  } else {  
    console.log('mode changé');  
  }  
});
```

`utimes(path, atime, mtime, callback)` modifie les dates de dernier accès (`atime`) et de modification (`mtime`) d'un fichier ou dossier. La logique est identique à toutes les autres fonctions présentées

# Manipuler des fichiers

- Lire le contenu d'un fichier : **readFile()**

```
fs.readFile(path, function (error, content) {  
    // Convertit le tampon de données en chaîne.  
    content = content.toString();  
  
    if (error) {  
        console.error('échec de lecture', error);  
    } else {  
        console.log('contenu du fichier', content);  
    }  
});
```

Pour lire le contenu d'un fichier, c'est tout naturellement que l'on se dirige vers `readFile(path, callback)`. Avant d'afficher son contenu (par exemple sur la console), il faut convertir le résultat en chaîne de caractères, d'où le `toString()`.

# Manipuler des fichiers

- Ecrire dans un fichier : **writeFile()**

```
fs.writeFile(path, content, function (error) {  
  if (error) {  
    console.error('échec de l\'écriture', error);  
  } else {  
    console.log('fichier écrit');  
  }  
})
```

L'écriture dans un fichier passe par la fonction `writeFile(path, content, callback)`. Cela reste toujours très simple d'utilisation : en paramètres, le chemin vers le fichier à écrire, le contenu (une chaîne de caractères ou un buffer) et toujours la callback.

# Manipuler des fichiers

- Supprimer un fichier : **unlink()**

```
fs.writeFile(path, content, function (error) {  
  if (error) {  
    console.error('échec de l\'écriture', error);  
  } else {  
    console.log('fichier écrit');  
  }  
})
```

La suppression d'un fichier se fait avec la méthode `unlink(path, callback)`, prenant en paramètres le chemin du fichier à détruire et la callback.

# Surveillance

- Les fonctions **watch**(path), **watchFile**(path) et **unwatchFile**(path) permettent de surveiller les changements d'un fichier ou d'un dossier.
- Le comportement de ces fonctions n'est pas consistant suivant les plates-formes
- Il est fortement conseillé d'utiliser le module gaze (<http://npmjs.org/package/gaze>)



# Surveillance

- Exemple de surveillance avec ***gaze***

```
var gaze = require('gaze');

gaze('*', { cwd: '/tmp' }, function (error) {
  if (error) {
    console.error('échec de la mise en place de la surveillance');
    console.error(error);
    return;
  }

  this.on('added', function (path) {
    console.log('nouveau fichier', path);
  });

  this.on('deleted', function (path) {
    console.log('fichier supprimé', path);
  });

  this.on('changed', function (path) {
    console.log('fichier modifié', path);
  });

  this.on('renamed', function (oldPath, newPath) {
    console.log('fichier renommé', oldPath, newPath);
  });
});
```