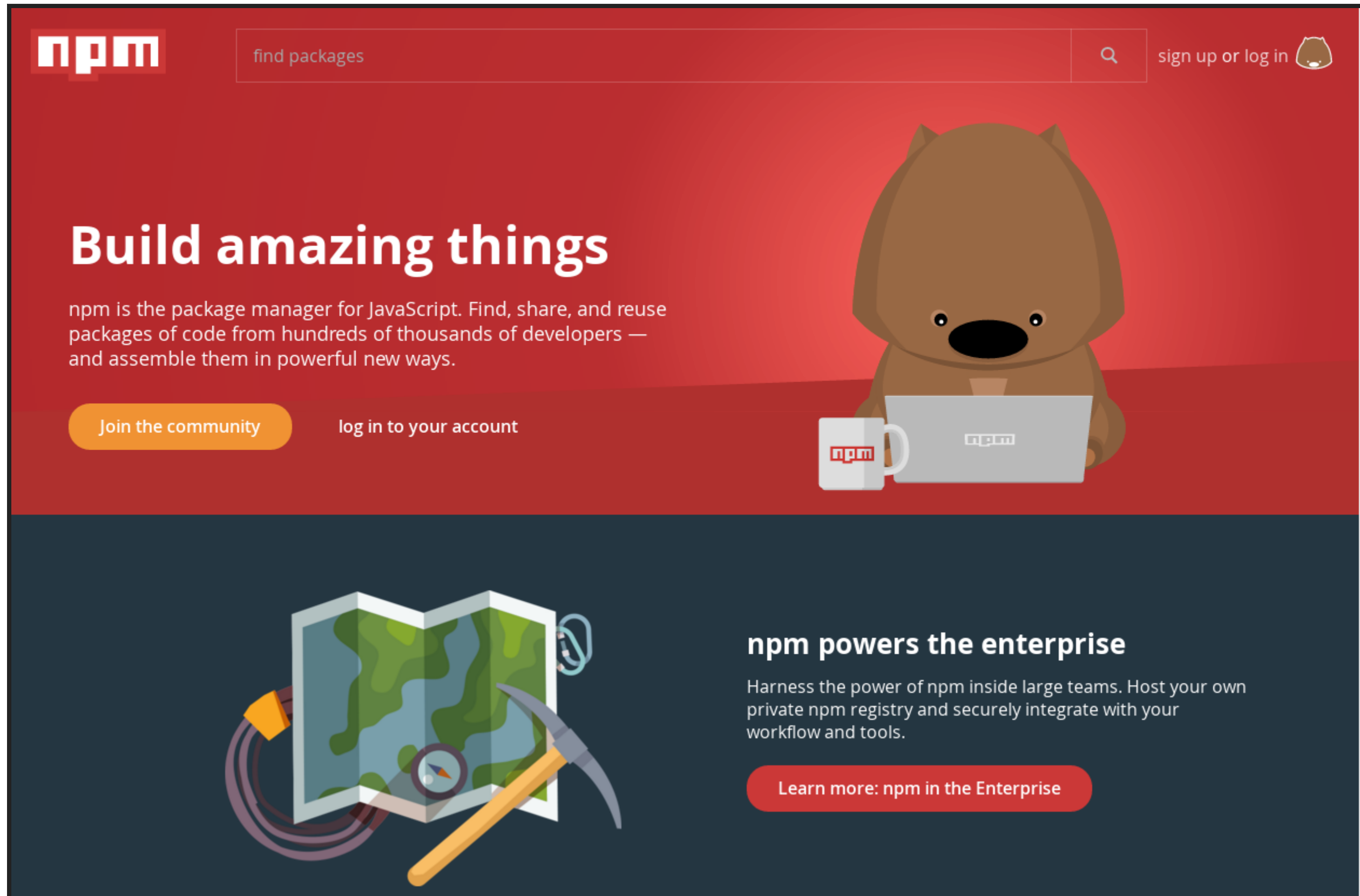


# PAKETMANAGEMENT & BUILDKRIPTE MIT NPM

# NPM IST EIN PACKAGE MANAGER FÜR JAVASCRIPT (NODE, FRONTEND) UND VERWALTET DIE ABHÄNGIGKEITEN EINER ANWENDUNG.



The image is a screenshot of the npm website homepage. The top section has a red background. On the left is the 'npm' logo. In the center is a search bar with the placeholder text 'find packages'. On the right is a search icon and a link 'sign up or log in' with a small cat icon. Below the search bar, on the left, is the heading 'Build amazing things' followed by a paragraph: 'npm is the package manager for JavaScript. Find, share, and reuse packages of code from hundreds of thousands of developers — and assemble them in powerful new ways.' Below this are two buttons: 'Join the community' (orange) and 'log in to your account' (white). On the right is a large illustration of a brown cat sitting at a desk with a laptop and a mug, both with the 'npm' logo. The bottom section has a dark blue background. On the left is an illustration of a map, a compass, and a pickaxe. On the right is the heading 'npm powers the enterprise' followed by a paragraph: 'Harness the power of npm inside large teams. Host your own private npm registry and securely integrate with your workflow and tools.' Below this is a red button with the text 'Learn more: npm in the Enterprise'.

**npm**

find packages

sign up or log in

## Build amazing things

npm is the package manager for JavaScript. Find, share, and reuse packages of code from hundreds of thousands of developers — and assemble them in powerful new ways.

[Join the community](#)

[log in to your account](#)

## npm powers the enterprise

Harness the power of npm inside large teams. Host your own private npm registry and securely integrate with your workflow and tools.

[Learn more: npm in the Enterprise](#)

**WAS IST EIN PACKAGE MANAGER?**

**WAS SIND KRITISCHE ASPEKTE?**

# NPM INFRASTRUKTUR

- Packages sind in zentraler Registry <https://www.npmjs.com/>
- Kommandozeilentool, ist in node enthalten
- Lokaler Cache in `~/ .npm` und `./node-modules`

ck@ck-travel ~

\$ npm

Usage: npm <command>

where <command> is one of:

access, add-user, adduser, apihelp, author, bin, bugs, c,  
cache, completion, config, ddp, dedupe, deprecate, dist-tag,  
dist-tags, docs, edit, explore, faq, find, find-dupes, get,  
help, help-search, home, i, info, init, install, issues, la,  
link, list, ll, ln, login, logout, ls, outdated, owner,  
pack, ping, prefix, prune, publish, r, rb, rebuild, remove,  
repo, restart, rm, root, run-script, s, se, search, set,  
show, shrinkwrap, star, stars, start, stop, t, tag, team,  
test, tst, un, uninstall, unlink, unpublish, unstar, up,  
update, upgrade, v, verison, version, view, whoami

npm <cmd> -h      quick help on <cmd>  
npm -l            display full usage info  
npm faq           commonly asked questions  
npm help <term>   search for help on <term>  
npm help npm      involved overview

Specify configs in the ini-formatted file:

/home/ck/.npmrc

or on the command line via: npm <command> --key value

Config info can be viewed via: npm help config

npm@3.3.12 /home/ck/.nvm/versions/v5.1.1/lib/node\_modules/npm

ck@ck-travel ~

# WICHTIGE BEFEHLE

- npm view ...
- npm install ...
- npm uninstall ...
- npm update
- npm outdated
- npm help

# ERSTELLEN VON NODE PACKAGES

- `npm init`
- Erstellt eine `package.json` mit der NPM Konfiguration



# BEISPIEL PACKAGE.JSON

```
{
  "name": "myproject-electron",
  "version": "1.0.0",
  "description": "Electron wrapper for MyProject",
  "main": "main.js",
  "scripts": {
    "start": "./node_modules/.bin/electron main.js",
    "pack": "./pack.js"
  },
  "author": "me",
  "devDependencies": {
    "electron-packager": "~6.0.0",
    "electron-prebuilt": "~0.37.0",
    "electron-winstaller": "~2.0.5",
    "minimist": "~1.2.0"
  },
  "dependencies": {
    "portfinder": "~1.0.3",
    "tree-kill": "~1.0.0"
  }
}
```

# INSTALLATION VON MODULEN

- Externe Abhängigkeiten werden mit `npm install <packagename>` installiert
- Mit `npm install --save <packagename>` wird die Abhängigkeit auch zur `package.json` hinzugefügt.
- Installierte Module landen im `node_modules` Verzeichnis (lokale Installation)
  - In `.gitignore` eintragen
- Pakete können auch global installiert werden: `npm install -g <packagename>`
- `PATH=./node_modules/.bin:$PATH`

# BROWSERIFY

- Verwendung von npm Modulen im Browser
- Baut aus mehreren npm Modulen eine einzige JS Datei
- Module haben ihren eigenen Namespace
- Verwendung von common.js (require/module.export)
- Module die auf node API (z.B. fs) Zugreifen können nicht benutzt werden. Browserify "ignoriert" `require( ' fs ' )` und gibt ein leeres Objekt zurück. Das require liefert somit noch keinen Fehler. Jedoch bei der Verwendung von fs würde es zu einem Laufzeitfehler kommen, da diese API nicht existiert. babelify (<https://www.npmjs.com/package/babelify>) browserify-shim Alternative: Webpack

**NPM KANN AUCH ZUM ERSTELLEN VON BUILD  
SKRIPTEN VERWENDET WERDEN.**

**WOZU BENÖTIGT MAN IN JAVASCRIPT PROJEKTEN  
BUILD SKRIPTE?**

# BUILD SKRIPTE

- `script` Abschnitt in der `package.json`
- Für vordefinierte Befehle (`npm start`, `npm test`) wird festgelegt welches Kommando ausgeführt wird
- Eigene Befehle können definiert werden und mit `npm run mycommand` ausgeführt werden
- In den Skripten sind i.d.R. die installierten Module ohne `node_modules/.bin` verwendet werden

```
"scripts": {  
  "start": "babel-node main.js",  
  "test": "mocha",  
  "mycommand": "echo hello"  
},
```

# ÜBUNG

- In `uebung_npm` existiert bereits ein `main.js`, dies startet einen einfachen HTTP Server.
- Erstelle ein npm Projekt und konfiguriere die `package.json` so dass mit `npm start` der Server gestartet wird.

TESTEN



# TEST FRAMEWORKS

- Es gibt unzählige Test Frameworks und Assertion Libraries
- Behandelt werden hier
  - Mocha Test Framework (<https://mochajs.org/>)
  - Chai Assertion Library (<http://chaijs.com/>)
  - Istanbul Code Coverage (<https://github.com/gotwarlost/istanbul>)
  - ESLint (<http://eslint.org/>)

# MOCHA

- Weit genutztes Test Framework
- Läuft im Browser oder auf der Konsole
- Unterstützt verschiedene Assertion Libraries

```
describe('Calculator', function(){  
  it('should add positive numbers', function(){  
    // test code  
  });  
});
```

```
$ npm install mocha  
$ node_modules/.bin/mocha  
Calculator  
  ✓ should add positive numbers  
  
1 passing (10ms)
```

# CHAI

- Assertion Library
- Unterstützt verschiedene Stile: should, expect, assert
- Prüfung von Werten, Arrays oder Objekten

```
var assert = require('chai').assert
assert.equal(3, 3);
assert.lengthOf([1, 2, 3, 4], 4);
assert.property({foo: 1}, 'foo');
```

# ISTANBUL

- Messen von Code Coverage
- Code wird on-the-fly instrumentiert
  - Es ist kein zusätzlicher Schritt im Build notwendig
- Verschiedene Report Formate

```
$ npm install istanbul
$ node_modules/.bin/istanbul cover node_modules/.bin/_mocha
...
===== Coverage summary =====
Statements   : 83.33% ( 5/6 )
Branches     : 100% ( 0/0 )
Functions    : 50% ( 1/2 )
Lines        : 83.33% ( 5/6 )
=====

$ firefox coverage/lcov-report/index.html
```

# ESLINT

- Linter für JavaScript/EcmaScript
- Erweiterbar mit Regeln und Plugins
- Achtung: Nach Installation sind alle Regeln deaktiviert!
  - Initiale Konfiguration erstellen: `eslint --init`
  - Konfiguration wird in `.eslintrc` oder in der `package.json` gespeichert

```
$ npm install eslint
$ node_modules/.bin/eslint --init
$ node_modules/.bin/eslint lib
```

# BEISPIEL (1)

Abhängigkeiten in der `package.json` konfigurieren:

```
"devDependencies": {  
  "chai": "^3.5.0",  
  "eslint": "^2.8.0",  
  "istanbul": "^0.4.3",  
  "mocha": "^2.4.5"  
}
```

## BEISPIEL (2)

ESlint in der `package.json` konfigurieren:

- Die empfohlenen Regeln aktivieren
- Mitteilen dass es sich um ein node.js Projekt handelt, damit z.B. `export` als globale Variable erkannt wird

```
"eslintConfig": {  
  "extends": "eslint:recommended",  
  "env": {  
    "node": true  
  }  
}
```

## BEISPIEL (2)

Das `npm test` Kommando konfigurieren:

- Mit `pretest` wird vor den eigentlichen Tests ESLint ausgeführt
- Mit `test` wird Mocha als Test Framework festgelegt
- Mit `posttest` wird nach den Tests noch die Code Coverage gemessen
  - Hier werden die Tests noch ein 2. Mal ausgeführt

```
"scripts": {  
  "pretest": "eslint lib",  
  "test": "mocha",  
  "posttest": "istanbul cover _mocha -- -u exports -R spec"  
}
```



# ÜBUNG

- Rufe `npm install` und `npm test` auf
- Korrigiere die von ESLint gefundenen Fehler
- Erhöhe die Test Coverage auf 100%

# MODULARISIERUNG

**WOZU BENÖTIGT MAN MODULE?**

# MÖGLICHKEITEN

## ES 5 - KEIN MODULKONZEPT

- ES5 Module Pattern
- CommonJS
- (AMD)

## ES 6 - NATIVE MODULE

# ES5 MODULE PATTERN

- Design Pattern
- Beispiel:

```
var calculator = (function () {  
    return {  
        add: function (a, b) {  
            return a + b;  
        }  
    };  
})();  
  
calculator.add(1, 2);
```

# COMMONJS

- Standard bei node.js
- im Browser z.B. mit Browsersify nutzbar
- Beispiel:

Moduldefinition in `calc.js`:

```
exports.add = function(a, b) {  
    return a + b;  
}
```

Import und Benutzung:

```
var calculator = require('./calc.js')  
calculator.add(1, 2);
```

# ES6 MODULE

- native Module
  - werden bisher weder von Browsern noch von node.js unterstützt
  - Übersetzung mittels Babel
- Beispiel:

Definition in `calculator.js`:

```
export function add (a, b) {  
  return a + b;  
}
```

Import und Benutzung:

```
import { add } from './calculator.js';  
add(1, 2);
```

# ÜBUNG

- Rufe `npm install` und `npm test` auf
- Refaktoriere `main.js`, extrahiere die Funktionen `add()` und `sub()` in ein ES6 Modul und binde dieses Modul dann in die `main.js` ein.



# FAZIT

Was habt ihr in diesem Kapitel gelernt?

**WAS WAR DAS WICHTIGSTE DAS  
DU HEUTE GELERN HAST?**

# SEITEN ZUM WEITERLERNEN

[OpenTechSchool](#)

[NodeSchool](#)

[ES6 Katas](#)

[Codecademy](#)

# BÜCHER ZUM WEITERLERNEN

[Speaking JavaScript](#)

[Exploring ES6](#)

[You Don't Know JS](#)