

# 2

## DEVELOPMENT FRAMEWORK NODE.JS, NPM, GIT

Salvatore Rinzivillo

# OBJECTIVES

- Setup a developing environment
  - Install Node.js and NPM
  - Install and configure Vue and vue-cli
  - Configure and initialize a project
- Install and configure git
  - Create a repository and import project files
- IDEs
  - Fork, Git Desktop
  - WebStorm, Atom.io, Textmate

# NODE.JS AND NPM

# WHAT IS NODE.JS

- “An asynchronous event driven Javascript runtime”
  - Non-blocking, event-driven I/O operations
  - Lightweight and efficient for data-intensive applications
  - Distributed computation and load balancing
- Available for download at <https://nodejs.org/>

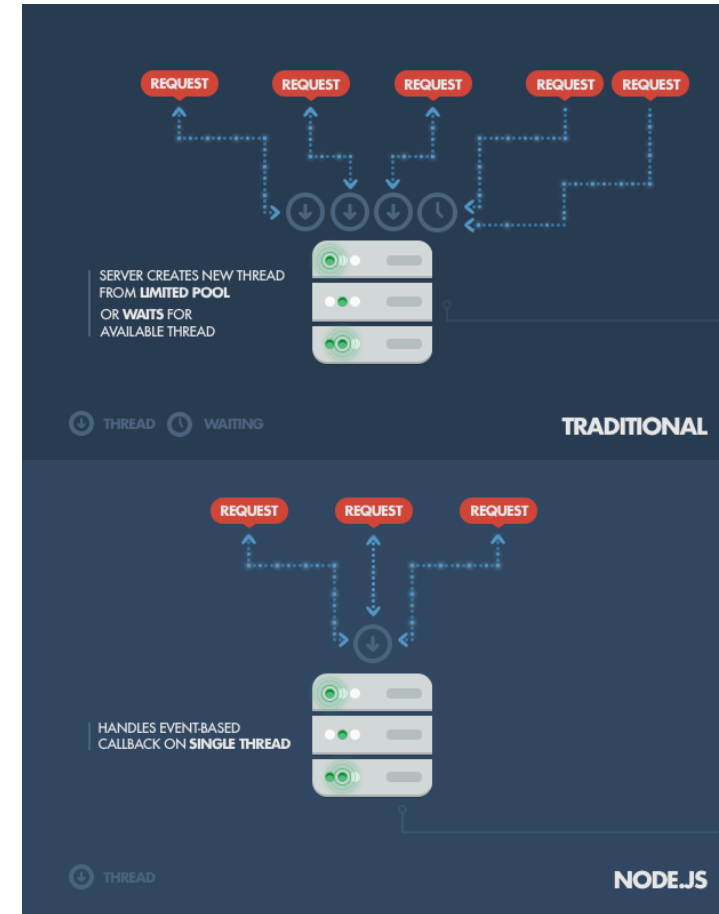


Image source: <https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js>

# NPM – NODE PACKAGE MANAGER

- Node.js has a large library of public available, reusable components
- Components are available through a repository
- Manage libraries for global use and local projects
- Handle all dependencies

# NPM - COMMANDS

- `npm init`
  - Initialize a project, creating a file `package.json`
- `npm install <module>`
  - Download and install module within the directory `node_modules`
  - With the flag `--save`, add the module to the `package.json` list of dependencies
  - With the flag `--global` (or `-g`) the module is installed globally on the system

# MOST USED PACKAGES

- Express: a web application development framework for node.js
- Lodash: general utilities for handling data structures in javascript
- http-server
- Specifically for the course:
  - D3
  - Plotly.js
  - Bootstrap
  - Vue.js
  - Vega; Vega-lite
  - ...

# EXERCISE — CREATE A PROJECT WITH NODE.JS

- Demo

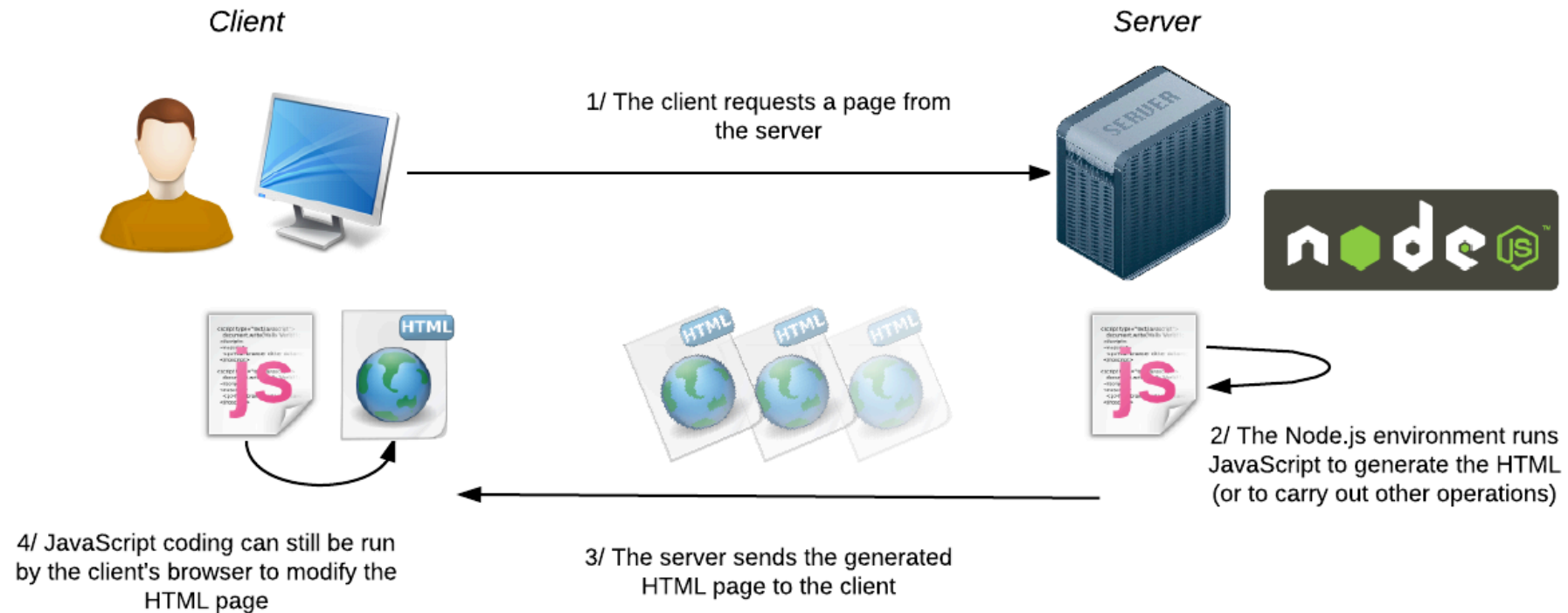


# CONFIGURING VUE.JS AND VUE-CLI

- `npm install -g vue-cli`
  - Create a command to manage Vue.js projects
- `vue init simple my-project`
- `cd my-project`
- `npm install`
- `npm run dev`
- These commands create a skeleton project configured with Vue.js

# WEB SERVER

# WEB SERVER FOR NODE.JS



# WEB SERVER IN NODE.JS

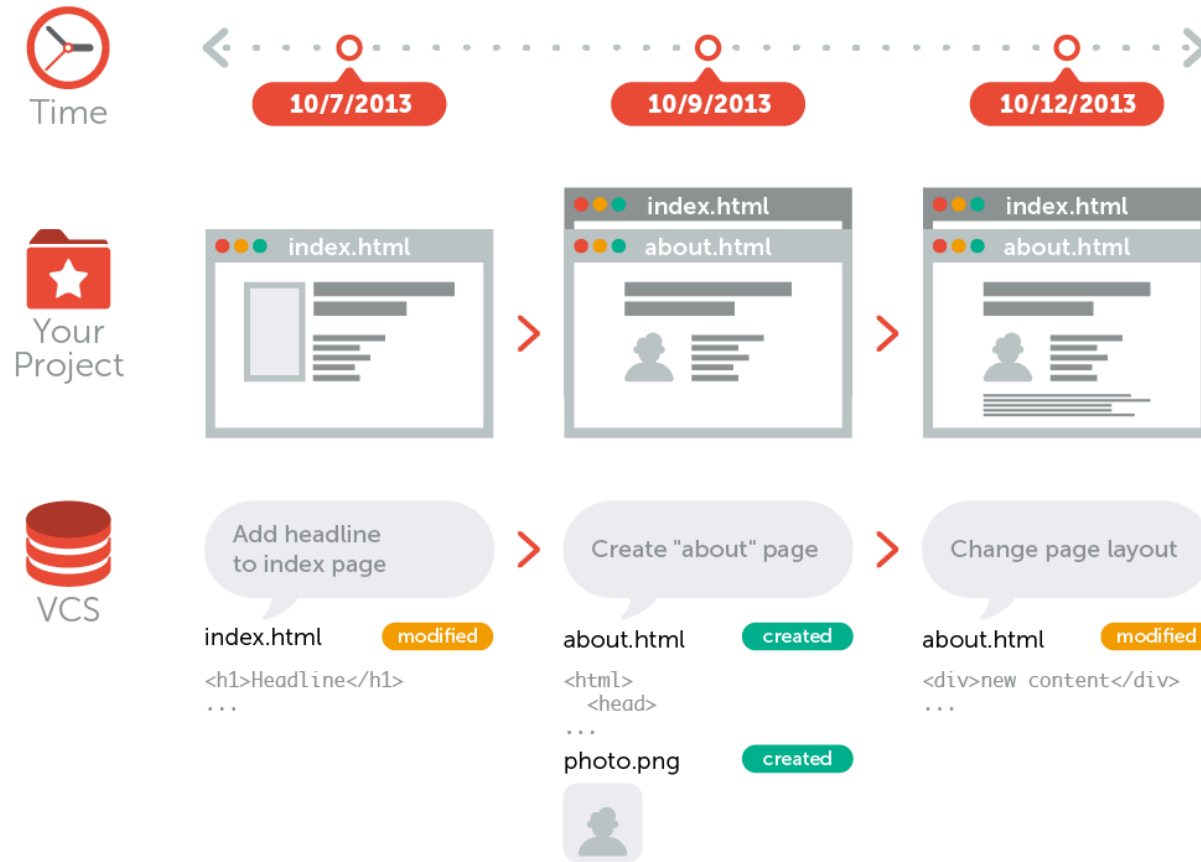
- There are several modules available for running a web server
- A very simple http server:
  - `npm install -g http-server`
- A more sophisticated application server:
  - `npm install -g express`

# EXERCISE — USE HTTP-SERVER TO ACCESS OUR PROJECT

- Demo

# VERSION CONTROL WITH GIT

# WHAT IS VERSION CONTROL?





# WHY USE A VERSION CONTROL SYSTEM?

- Collaboration
  - Any member of a team can work on any file at any time
  - Merge of contribution is handle by the VCS
- Storing versions
  - Tracking of changes through periodic saves of snapshots
  - Only one version of a project at any time
    - Other versions are packed within the VCS
- Restore previous versions
- Follow the development of the project
- Backup, when using external repositories

# WHICH VCS? INTRODUCING GIT

- Download a client from public repositories
  - For example: GitHub, BitBucket
- Use clients specific for your OS
  - For example: `brew install git` (for MacOSX)
- Initial configuration
  - `git config --global user.name "rinziv"`
  - `git config --global user.email "rinzivillo@isti.cnr.it"`
  - `git config --global`

# GIT – CREATING A REPOSITORY

- GIT handles two kinds of repositories
  - Local repository
    - Contained within a folder `.git` in the root of the project folder
    - Only on person access this repo
  - Remote repository
    - Located into a remote server
    - Locally stored within the `.git` folder
    - Team members work concurrently on remote repository

# GIT – CREATE A LOCAL REPOSITORY

- Move within the project root directory
- Use `git init` to start versioning tracking
- The root of the project is called `working copy`
  - There is only one working copy at any moment
  - It is possible to update the current working copy with previous versions from the repository
- Some files (usually related to the OS) can be ignored for the versioning
  - Create a file called `.gitignore` in the root of the project folder
  - List the files to ignore within the file

# GIT – CLONE A REMOTE REPOSITORY

- A remote repository have a URL of the form:
  - `ssh://user@server/git-repo.git`
  - `user@server:git-repo.git`
  - `http://example.com/git-repo.git`
  - `https://example.com/git-repo.git`
  - `git://example.com/git-repo.git`

# GIT - COMMIT

- Commit operation save the snapshot of the working copy on the repository
  - `git add -A`
  - `git commit -m "Initial commit"`

# GIT – STATUS OF THE PROJECT

- Each file within the project have one of the following state
  - **Untracked**: the file is not under version control. GIT do not track any change on this file
  - **Tracked**: GIT reports changes on these files
- **git status** reports the list of files within the project that have changed and those that are not tracked

# GIT – STAGING AREA

## Working Copy

Your Project's Files



Git watches tracked files for new local modifications...

## Staging Area

Changes included in the Next Commit

## Local Repository

The ".git" Folder

### Tracked (and modified)



If a file was modified since it was last committed, you can stage & commit these changes

stage



Changes that were added to the Staging Area will be included in the next commit

commit



All changes contained in a commit are saved in the local repository as a new revision



Changes that are **not staged** will not be committed & remain as local changes until you stage & commit or discard them

### Untracked



Changes in untracked files aren't watched. If you want them included in version control, you have to tell Git to start tracking them. If not, you should consider ignoring them.



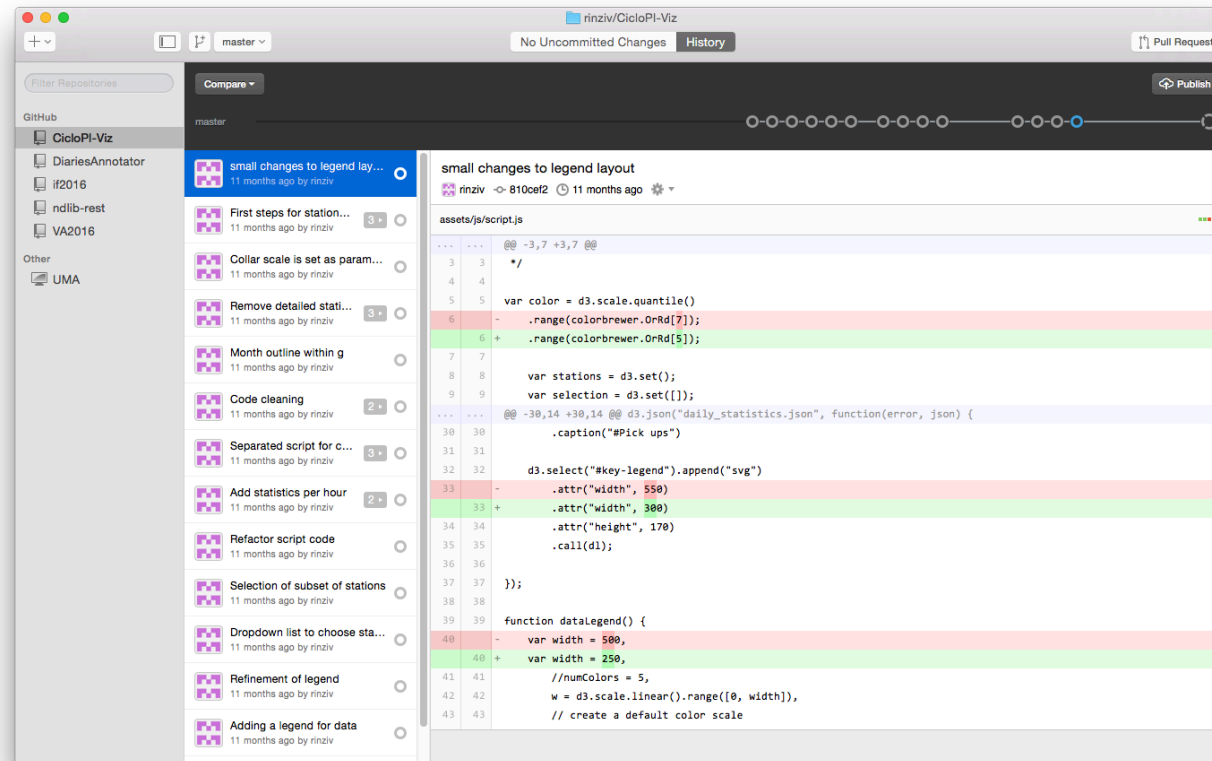
# GIT – PREPARING THE STAGING AREA AND COMMIT

- Add the files to include in the staging area
  - `git add new-page.html index.html css/*`
  - `git rm error.html`
  - Check the status with: `git status`
- Commit changes
  - `git commit -m "Implement the new login box"`

# WHEN TO COMMIT?

- Each commit should contains changes related to a single topic
- Save only completed work (for temporary saving use Stash)
- Test the project before committing
- Add descriptive message
- Commit often

# DESKTOP GUI VERSION



# FORK

The screenshot displays a GitHub repository for 'swift' with the 'master' branch selected. The interface is divided into three main sections: a left sidebar, a central commit history, and a right sidebar.

**Left Sidebar:** Shows repository navigation options including 'Changes', 'All Commits', 'Starred', and 'Branches'. Under 'Branches', 'master' is selected, and other branches like 'origin', 'gsb-superclass', 'master-next', 'swift-5.0-branch', and 'test' are listed. The 'Remotes' section shows 'origin' with various sub-branches.

**Central Commit History:** A list of commits is shown, with the most recent one highlighted in blue. The commit message is '[ParseableInterface] Ensure module-link-name is serialized and used'. The commit is attributed to Harlan Haskins (HH) and is a merge pull request from harlanhaskins/the-missing-link. The commit SHA is 2889a1f906c481df1037b718bbb3b6289a8d6a27.

**Right Sidebar:** Shows a list of recent commits with their authors, commit IDs, and timestamps. The top commit is by Harlan Haskins (HH) with ID 2889a1f, dated 28 Nov 2018 at 08:12.

**Bottom Section:** Displays the commit details for the selected commit, including the author's name (Harlan Haskins), email (harlan@apple.com), and the commit message. Below this, there is a section for the merge pull request, showing the source branch and the commit message. At the bottom, there is a file tree view showing the file 'lib/Frontend/ParseableInterfaceSupport.cpp'.

# EXERCISE – CREATE A REPOSITORY FOR OUR PROJECT

- Demo

# CLASSROOM SPRING 2020 REPOSITORY

<https://github.com/va602aa-2021>

