

Salvatore Rinzivillo

VISUAL ANALYTICS



DEVELOPMENT FRAMEWORK

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/>

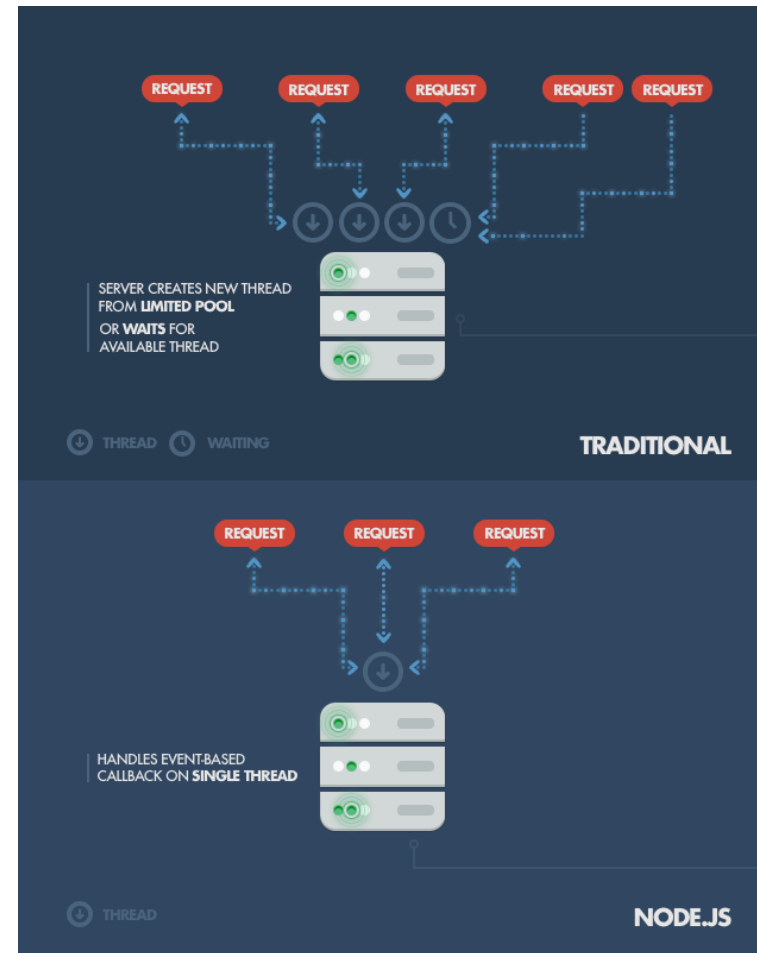


Image source: <https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js>

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
 - ...

Exercise – Create a project with Node.js

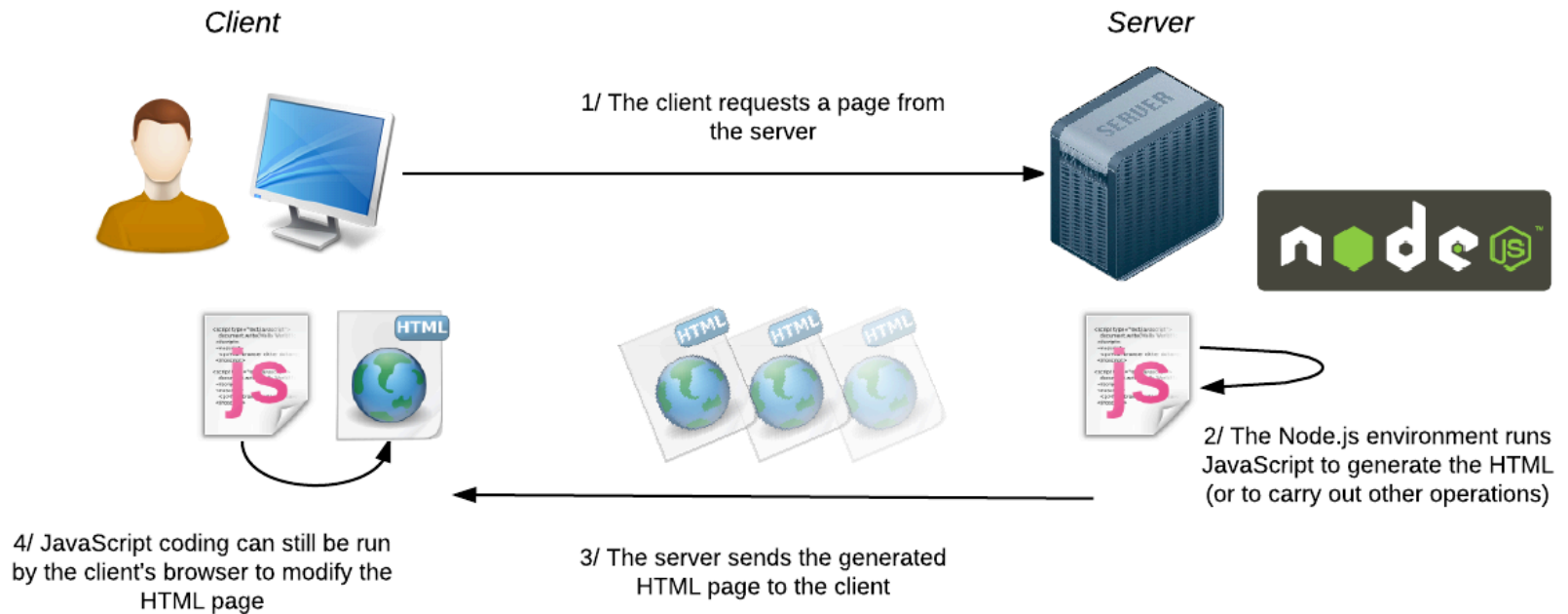
- Demo

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

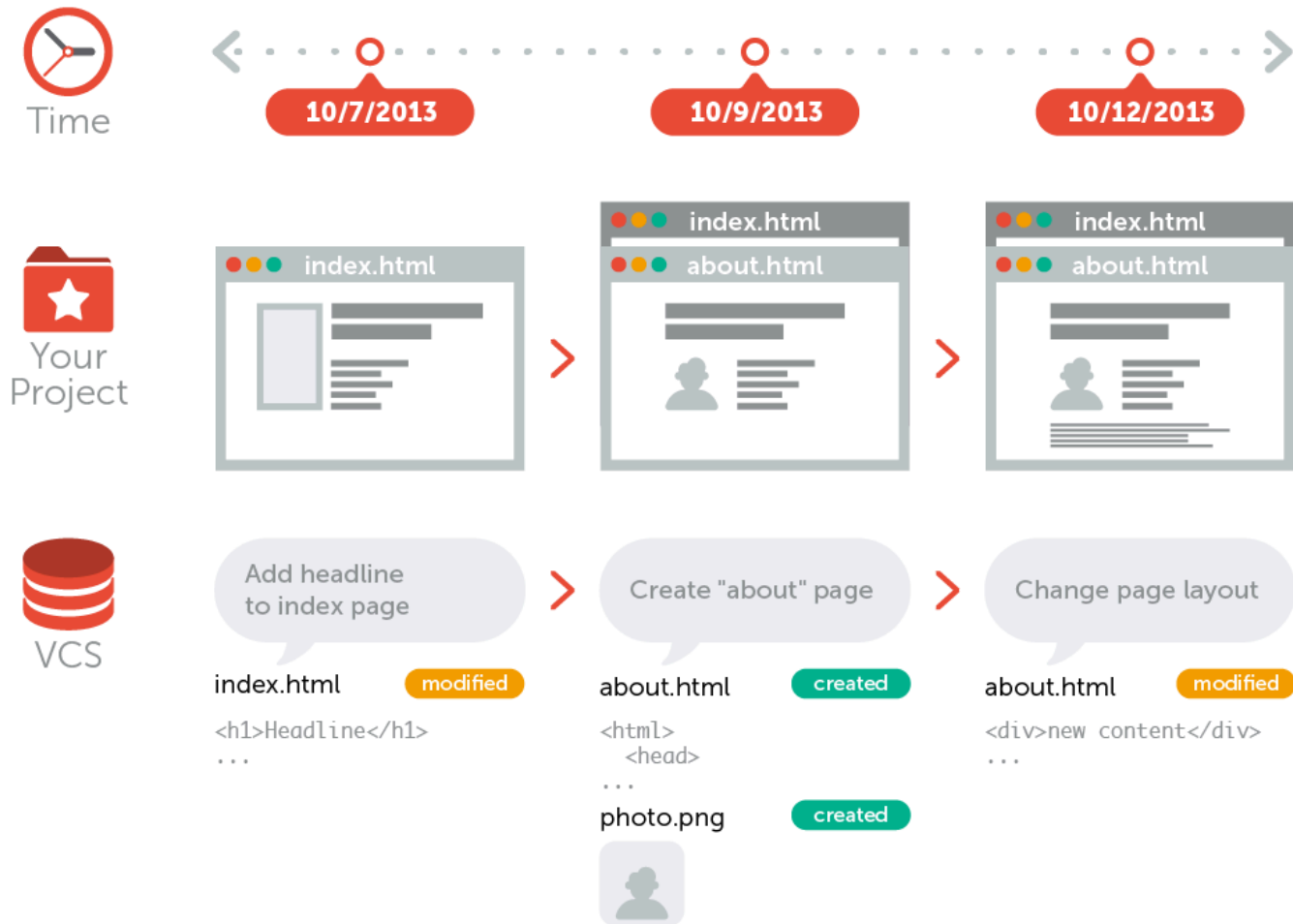
- There are several modules available for running a web server
- A very simple http server:
 - `npm install -g http-server`
- A more sophisticated application server:
 - `npm install -g express`

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 one 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...

Tracked (and modified)



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

stage



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.

Staging Area

Changes included in the Next Commit



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

commit

Local Repository

The ".git" Folder



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

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

The screenshot displays a desktop application window titled "rinziv/CicloPI-Viz". The interface includes a top bar with "No Uncommitted Changes" and "History" buttons, and a "Pull Request" button on the right. A left sidebar shows a "Filter Repositories" input and a list of repositories under "GitHub" (CicloPI-Viz, DiariesAnnotator, if2016, ndlib-rest, VA2016) and "Other" (UMA). The main area is split into two panes. The left pane, titled "Compare", shows a list of commit messages from "rinziv" dated "11 months ago", such as "small changes to legend lay...", "First steps for station...", "Collar scale is set as param...", "Remove detailed stati...", "Month outline within g", "Code cleaning", "Separated script for c...", "Add statistics per hour", "Refactor script code", "Selection of subset of stations", "Dropdown list to choose sta...", "Refinement of legend", and "Adding a legend for data". The right pane shows a diff view for the commit "small changes to legend layout" (810cef2). The file being edited is "assets/js/script.js". The diff highlights changes in red (removal) and green (addition). The code includes a D3.js legend function that uses colorbrewer scales and sets attributes for width and height.

```
assets/js/script.js
@@ -3,7 +3,7 @@
3 3  */
4 4
5 5  var color = d3.scale.quantile()
6 6  - .range(colorbrewer.OrRd[7]);
7 7  + .range(colorbrewer.OrRd[5]);
8 8
9 9  var stations = d3.set();
10 10  var selection = d3.set([]);
... .. @@ -30,14 +30,14 @@ d3.json("daily_statistics.json", function(error, json) {
30 30  .caption("#Pick ups")
31 31
32 32  d3.select("#key-legend").append("svg")
33 33  - .attr("width", 550)
34 34  + .attr("width", 300)
35 35  .attr("height", 170)
36 36  .call(d1);
37 37  });
38 38
39 39  function dataLegend() {
40 40  - var width = 500,
41 41  + var width = 250,
42 42  //numColors = 5,
43 43  w = d3.scale.linear().range([0, width]),
// create a default color scale
```

Fork

The screenshot shows a Git client interface with a commit history on the left and a detailed view of a merge pull request on the right.

Commit History:

Commit Hash	Author	Message	Date
806cf57	swift-ci	Merge pull request #20825 from DougGre...	28 Nov 2018 at 09:36
da772e9	Doug Gregor	[Runtime] Make swift::swift_conformsToSwiftProtocol overridable.	28 Nov 2018 at 08:37
f9797b4	Doug Gregor	[Runtime] Add dummy "module name" parameter to swift::_conformsTo...	28 Nov 2018 at 01:11
5528db1	swift-ci	Merge remote-tracking branch 'origin/master' into master-next	28 Nov 2018 at 08:30
2889a1f	Harlan Haskins	Merge pull request #20816 from harlanhaskins/the-missing-link	28 Nov 2018 at 08:12
717532e	Harlan Haskins	[ParseableInterface] Test that module-link-name is serialized and used	28 Nov 2018 at 01:12
3e15224	Harlan Haskins	[ParseableInterface] Serialize module-link-name in binary module	28 Nov 2018 at 00:08
618f3a6	swift-ci	Merge remote-tracking branch 'origin/master' into master-next	28 Nov 2018 at 07:09
e4efac5	Saleem Abdulrasool	Merge pull request #20717 from trougton/patch-2	28 Nov 2018 at 07:04
89d6cdf	Thomas Roughton	Define mappings for CF types on LLP64	25 Nov 2018 at 08:00
0b68085	Saleem Abdulrasool	Merge pull request #20813 from drodriguez/fix-android-gold-link...	28 Nov 2018 at 07:00
def1172	Daniel Rodríguez Tro...	Split link flags and link libraries lists to make gold happy.	28 Nov 2018 at 00:24

Merge Pull Request #20816:

AUTHOR: Harlan Haskins (harlan@apple.com), 28 November 2018 at 08:12:57 CET

COMMITTER: GitHub (noreply@github.com), 28 November 2018 at 08:12:57 CET

SHA: 2889a1f906c481df1037b718bbb3b6289a8d6a27

PARENTS: e4efac5 717532e

Message: Merge pull request #20816 from harlanhaskins/the-missing-link

Commit Message: [ParseableInterface] Ensure module-link-name is serialized and used

File Tree: lib/Frontend/ParseableInterfaceSupport.cpp

<https://git-fork.com>

Exercise – Create a repository for our project

- Demo

Classroom Spring 2020 repository

<https://github.com/va602aa-2020>