Web Programming 04/18

* Git Basics
* Node.js Intro
* Node Package Manager (npm)
* Node Version Manager (nvm)
* Interactive nvm (n)

Just in case you haven't install…

- Git
  - https://git-scm.com/book/zh-tw/v2 (請安裝命令列的 git, 不要只是仰賴 GUI)
  - 影片：https://www.youtube.com/watch?v=HVsySz-h9r4
- Node.js
  - 教學：https://github.com/...../node/learning-node-module.md
  - 下載：https://nodejs.org/en/download/
- npm... 應該裝了 node 就已經裝了吧？
- nvm
  - https://github.com/creationix/nvm
- n
  - https://github.com/tj/n

Version Control Tools

- 顧名思義，就是幫你管理同一個檔案不同版本的工具，可以幫你記錄版本之間之不同 (diff)、每個版本的修改理由(log/comment)、快速回朔到先前版本、以及針對不同版本之間給予標記/版號 (tag) 等等。

- 此外，也由於這些工具對於版本之間只儲存差異的部分，因此，也可以大幅的節省儲存空間，並且達成更乾淨的檔案管理等好處。

- 當然，人們也常常利用版本控制工具來進行文件/程式的協作，讓不同的人可以在 local machine 備份(checkout)一份文件/程式 (aka. sandbox)，在 local 進行修改，而不會影響到 central 的版本，待個人編寫完畢、且通常經過測試無誤之後，再行上傳到 central (checkin)。而版本控制工具也因此常會提供一些像是 checkout/checkin、狀態檢查 (status)、merge、lock 等功能，以利不同人之間之協作。

Types of Version Control Tools

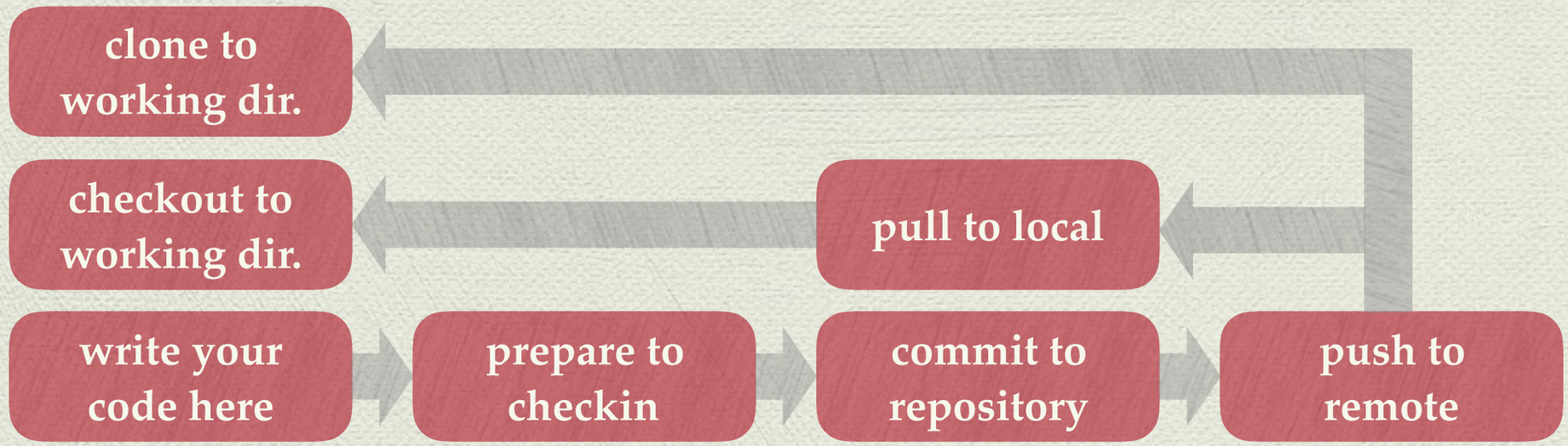
- **Local data model** — All developer must use the same file system
  - e.g. Revision Control System (RCS)

- **Client-server model** — Developers use a shared single repository
  - e.g. Concurrent Versions System (CVS)、Subversion (SVN)

- **Distributed model** — Each developer works directly with his or her own local repository, and changes are shared between repositories as a separate step
  - e.g. Git (designed by Linus Torvalds)

Ref: https://en.wikipedia.org/wiki/List_of_version_control_software

Git Basic Ideas



Stash

A place to hide modifications while you work onsomethingelse

Working Place

Local working place (directory)

Staging / Index

Files you "cache" before checking in (committing) to git repository

Local Repository

A local directory that contains all of the necessary repository files. The directory name is ".git".

Upstream / Remote Repository

Versions hosted on the internet, ensuring changes are available for other developers. The default name is "origin".

Initialize a Git project

- For a new or existing working (coding) directory, we want to add the Git version control over it.

- [Command] `git init`
  - A local repository ".git" is created
  - What's inside .git directory? What's the content of "HEAD"?

Basic Git file operations

- `git add` — Add (modified) file(s) to staging area
- `git commit [-m "message"]` — Commit the stored changes from staging area to local repository
  - What's inside .git directory now?
- `git status`
- `git diff`
- `git rm [-f]` — Remove file(s) from the working directory and staging area
  - The "-f" option is needed when the file is still in the working directory
- `git reset` — Reset current HEAD (branch head) to the specified state
  - Basically, it removes what have been put in staging and reset to <commit>
- `git log` — Show commit logs

Git Practice #1

- Open a new directory “git-test”. Change directory to it and execute “`git init`”
- Edit two files: “hello”, “world”.
- Do the following git commands:
  - `git status`
  - `git add hello`
  - `git commit -m "Init hello"`
- Modify “hello” again, and do the following git commands:
  - `git status`
  - `git add -A`
  - `git status`
  - `git commit`
- Modify “hello” and “world” again, and do the following git commands:
  - `git add hello`
  - `git rm hello` // Error! Why? How to fix it?

Git Practice #1 (continued)

- Do the following git commands:
  - `git rm -f hello`
  - `git status`
- In command line, do "`ls`". What do you see?
- Uh, the file "hello" is gone… What can you do to recover it?
- `git checkout` — Check out the committed version from .git repository
- Do the following git commands:
  - `git reset`
  - `git status`
  - `git checkout` // Does this work?
  - `git checkout hello`
  - `git status`
  - `git log`

Publish your local repository to Github

- Github — 全世界最大的軟體原始碼代管服務，使用 git 來進行版本控制
  - 其他比較知名的軟體原始碼代管服務包含 bitbucket (also git-based), SourceForge (支援各種 version control tools) 等

- [Method #1] Go to github. Log in to your account
  - 直接點選 “New Repository”, 取個名字 (e.g. “git-remote-test”)
  - In command line, do:
    - `git remote add origin https://github.com/<yourID>/git-remote-test.git`
    - `git push -u origin master`
- [Method #2] 下載 & 使用 github desktop： https://desktop.github.com/
  - File -> clone repository

Manage remote repository on Github

- Also called “upstream” repository
- `git remote` — Manage set of tracked repositories
  - Note: Each remote repository is “named”. The default name is “origin”.
  - [Options/Subcommands]
    - `-v` // Show remote information
    - `add <remoteName> <url>` // add a remote repo with url
      // e.g. `add origin https://github…`
    - `rm <remoteName>` // remove the remote named repo
      // e.g. `rm origin`
- `git push <remoteRepo> <srcBranch>` — Update remote repository
  - e.g. `git push -u origin master`
    // Push the committed changes in local “master” branch to remote “origin” repo.
    // The “-u” option sets the upstream repo so that you don't need to specify it
    // every time

Git Practice #2

- Create a new github repository. Push your local repo to it.
- What's the content in ".git" now?
- Do the following git commands:
  - `git remote`
  - `git remote -v`
  - `git commit -m "Init hello"`
- Modify "hello", commit to local repo, and publish (push) it to the remote repo
  - `git add hello`
  - `git commit`
  - `git push [origin]`
- Go to github and see if the remote "hello" has been changed

Clone a remote repository from Github

- It's often that you would like to "import" source codes from a remote repo
  - To use it as a utility/library/module
  - To improve/co-work on the codes

- `git clone <repoUrl> [<localDir>]` — Cloning the remote repo to the local working directory
  - What's the url for the remote repo you just created?

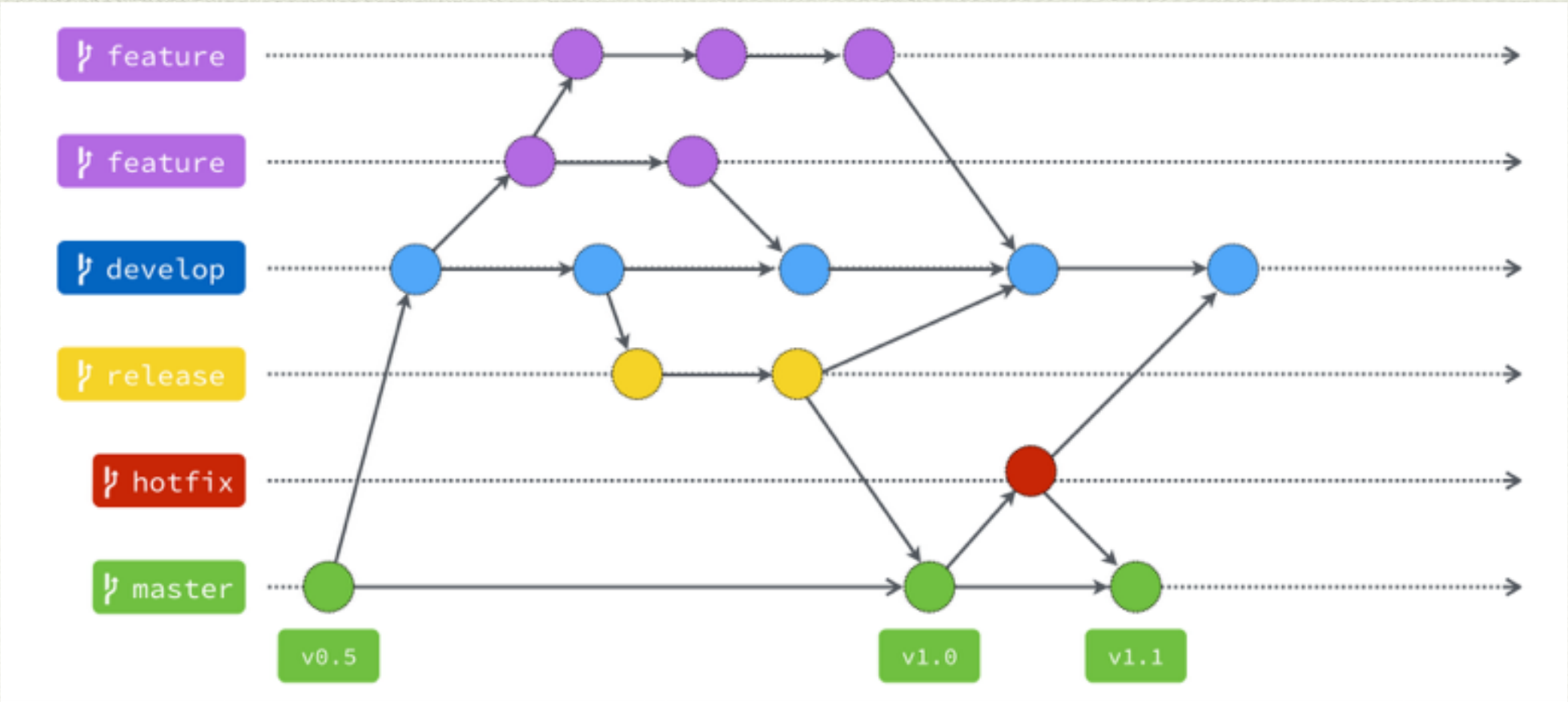
- `git pull <remoteRepo> <srcBranch>` — Update local repository
  - e.g. `git pull [origin master]`
    // Update the local "master" branch with the remote "origin" repo.
  - In its default mode, `git pull` is shorthand for `git fetch` followed by `git merge FETCH_HEAD`

Git Practice #3

- Find someone's remote repo. Clone it to your local directory.
  - If there is no "someone", then clone your own remote repo to another directory
- In your local repo, modify "hello", commit to local repo, and publish (push) it to the remote repo. Ask the other one (author) to update (pull) his/her local repo.
- In your local repo, modify "hello" again, and commit it to the local repo (i.e. your local master is newer then the remote origin).
  - What happens if you do "`git pull origin master`"? (Note: not "git push"!!)
- In the other author's local repo, modify "hello" differently (with yours). Commit it to his/her local repo and publish (push) to the remote origin.
  - What happens if you do "`git status`"?
  - What happens if you do "`git pull origin master`"?
  - What happens if you do "`git push origin master`"?
  - Try to modify files in different ways, and see how automatic merging behaves?

Git Flow / Git Branch

- In the previous practice, it happens that when different authors modify the same file differently at the same time, git update may fail.
  - Disasters may happen if everybody has different principles towards commits and code quality.
- Git Flow model — http://nvie.com/posts/a-successful-git-branching-model/



Create Git Branches

- 事實上所謂的 git flow 只是一種精神，不同的應用、不同的單位、都可能有自己的 git flow. 重點是用 git 的 branching and merging 系列的指令來管理個人(developer) 以及內部與外部 release 環境 (e.g. staging and production) 下不同分支 (branches) 底下的版本控制，以及如何合流(merge)以及管理版號等等 issues.

- `git branch` — List, create, or delete branches
  - `git branch <newBranchName> [<startPointTagName>]`
    - e.g. `git ranch newBdd v2.3.06`
    - Default <startPointTagName> is “HEAD”
  - `git branch -d <branchName>`
  - `git branch --list`
  - `git branch --merged` // show merged branches

- `git checkout` — Switch branches (or restore working tree files)

- `git tag <tagName>` — Create, list (--list), or delete (-d) a tag

- `git merge <branchName>` — Merge <branchName> to the checked out branch

Git Practice #4

- First, let's make the current version v0.1 and add a tag
  - `git tag v0.1` // check out the contents in .git
- Now, let's create a new branch (dev-0.2) based on v0.1
  - `git branch dev-0.2 v0.1` // => check out the contents in .git
  - `git branch` // still in master!!
  - `git checkout dev-0.2` // switch to "dev-0.2" branch
- Modify "hello", add and commit it. Now we would like to merge the modifications on branch "dev-0.2" back to master (v0.1)
  - `git checkout master` // switch to "master" branch
  - `git merge dev-0.2` // merge dev-0.2 with master

Git in Practice

- 從現在開始，請大家在寫作業的時候盡量用 git 來進行版本控制
  - 維持一個隨時可以 demo 的 master (適時的給一些 tags)
  - 如要開發一個新功能，就 new 一個 branch，並且在那邊完成開發與測試之後，再 checkin 回 master
- 如要與別人協作，就在 github 開一個協作的 repo, 然後各自 clone 回 local 進行開發
  - 為了維持 master 的可運作性，建議先從 master 開一個 develop (or staging) branch, 然後各自的 local repo 則由此 branch 展開，而各自完成的部分可以先 checkin 回這個 branch, 等到達成一個 milestone 且經過整合測試之後，再把這個 branch merge 回 master, 並打版好。
- 一些不錯的 git references:
  - Git command ref: https://git-scm.com/docs
  - Git cheatsheet: http://ndpsoftware.com/git-cheatsheet.html


node
JS
TM

前端 vs. 後端 (again)

- Recall: 前端(front-end)的程式(html)就是讓瀏覽器在讀入之後可以轉譯(interpret)成畫面，然後透過 Ajax 技術以及 script language (如：JavaScript) 的編寫，讓網頁使用者可以透過一些輸入裝置(如：鍵盤、滑鼠)進行互動。
- 但是，這樣的前端網頁程式最大的不足就是所有互動過的資訊，都會隨著瀏覽器的重新整理而不見，而無法做到讓使用者有像是會員資料之類的互動。
- 所以，後端(back-end)程式就是一個寫在伺服器的程式，可以提供資料的存取或是各種資源模組的運用。
- 因此，巨觀來說，任何可以跟瀏覽器透過網路協定 (如：http) 來溝通的程式語言都可以用來寫後端
  - 包含大家熟悉的：C/C++, Java, Python, Perl… 等, 以及針對網路服務而產生的：PhP, .Net, Ruby… 等
  - How about JavaScript?

JavaScript as a back-end language

- Recall: JavaScript 的誕生
  - In 1995, Netscape 與 Sun 合作，想要基於 Java 推出一個 script language 來讓使用者可以跟網頁互動
  - Brendan Eich 剛加入了 Netscape 不久就被指派了這個任務。但問題是他不喜歡 Java, 所以就花了 10 天內(based on C++, Scheme, Self) 設計出來一個用來交差的語言 JavaScript
- 先不說 JavaScript 各種混亂的語法所造成的困擾，它原本連 module 也沒有 support (ES6 以後才 support), 所以很難實作出堪用的後端程式
- 因此，Node.js 就是基於 JavaScript, 想要提供一個 server 端的執行環境的計畫

Ref: http://www.ruanyifeng.com/blog/2011/06/birth_of_javascript.html

Node.js 的誕生

- In 2009, Ryan Dahl (28 歲) 基於 Google Chrome 的 v8 engine, 開發了 node.js，希望可以打造用 JavaScript 來寫後端的環境
  - Google Chrome V8 engine
  - libuv — Cross-platform (*nix, Windows) asynchronous I/O library
  - Non-blocking, event-driven, single-threaded

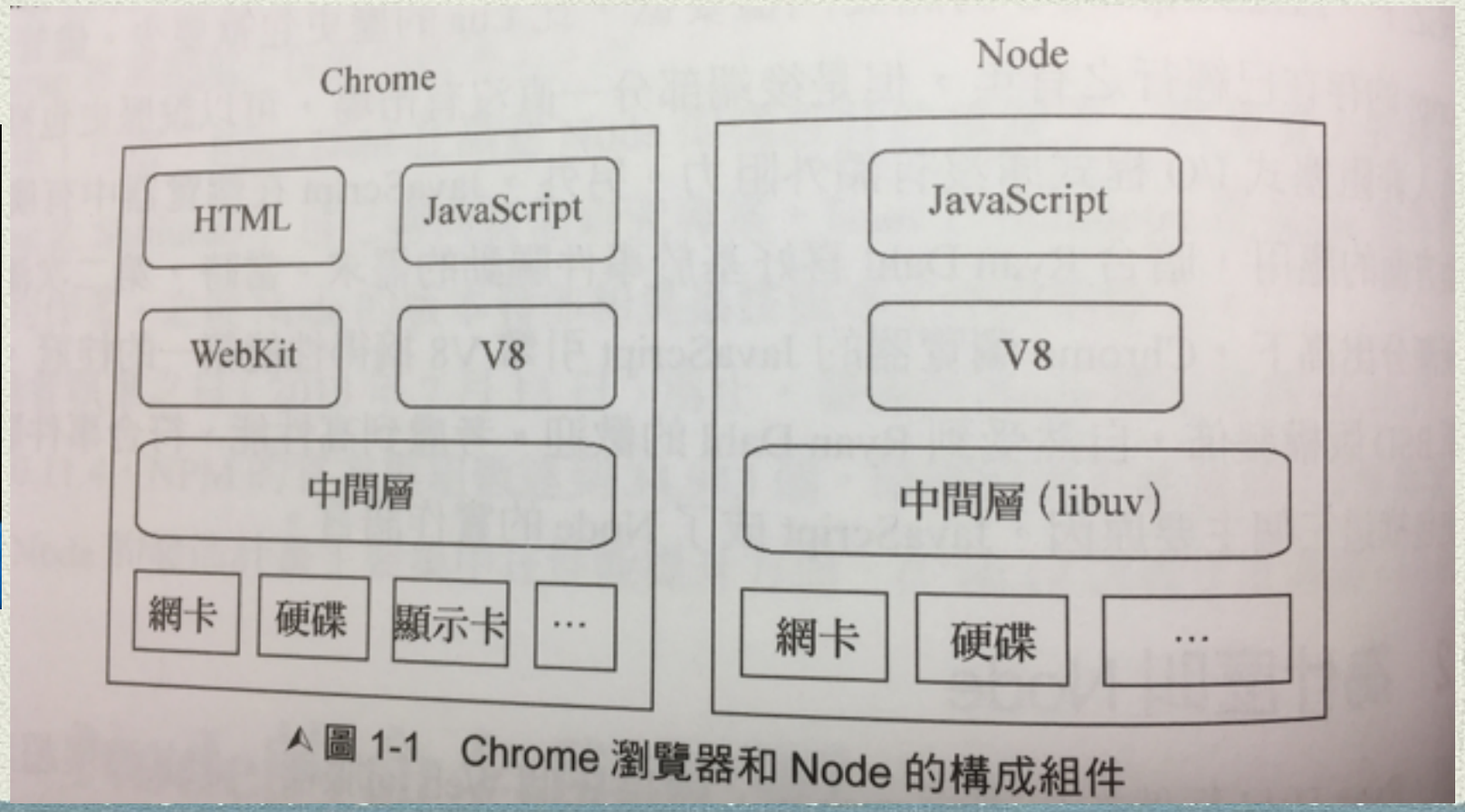
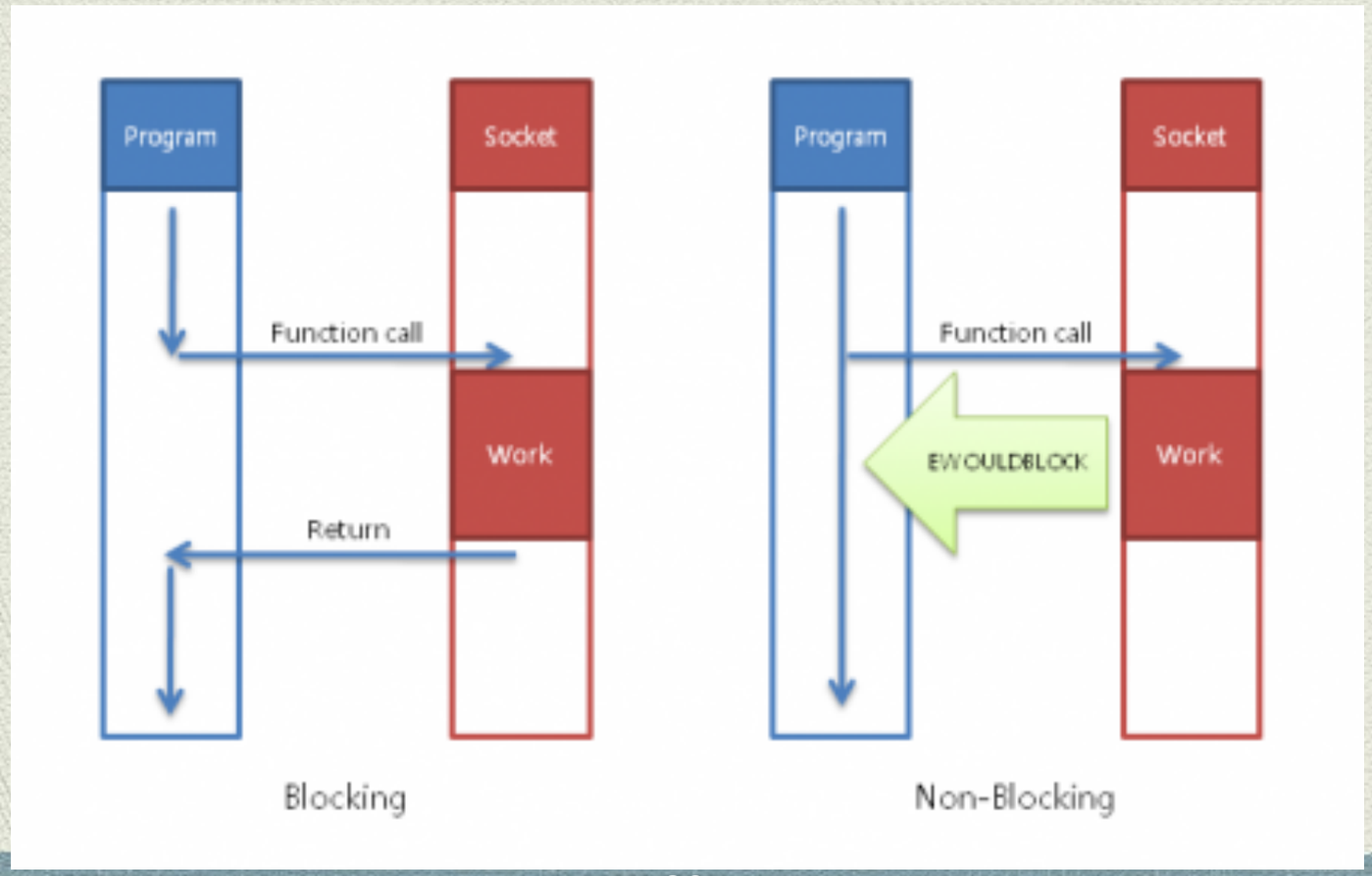


圖 1-1 Chrome 瀏覽器和 Node 的構成組件

Blocking vs. Non-blocking

- How PHP or ASP handles a file request:
  1. Sends the task to the computer's file system.
  2. Waits while the file system opens and reads the file.
  3. Returns the content to the client.
  4. Ready to handle the next request.

- How Node.js handles a file request:
  1. Sends the task to the computer's file system.
  2. Ready to handle the next request.
  3. When the file system has opened and read the file, the server returns the content to the client.



Node Modules

- Node.js 作為一個後端的環境，除了上述的特點之外，另外也很重要的是要又各種內建、以及能快速的擴充、載入的各種模組
- 然而 JavaScript 一直到 ES6 (2015) 年才正式 support modules, 因此，node.js 採用了 CommonJS 的模組規範，來支援一些用原生 C++ 撰寫的模組，像是網路協定 (http)、檔案系統 (fs)、資料庫、加密(crypto) 等功能
- 載入模組 — `require('模組名稱')`
  - `var http = require('http');  // 原生模組可以直接指定名稱`
  - `var fs = require('fs');`
  - `var express = require('express'); // npm install express;`
    `// 第三方模組也可以直接指定`
  - `var utils = require('../lib/utils');  // 或是給定相對路徑`
- 引入後就可以直接使用
  - ```
    const math = require('math');
    console.log(math.add(1, 2));
    ```
    ```

Node 內建 Modules

- Ref: https://nodejs.org/api/modules.html

Node.js v9.11.1 Documentation

Index | View on single page | View as JSON | View another version ▼

Table of Contents

Node http module

```
const http = require('http');
const PORT = process.env.PORT || 3000;
const server = http.createServer(function(req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World!');
});
server.listen(PORT, function() {
  console.log
  ('Server listening on: http://localhost:${PORT}');
});
```

Node fs module

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

fs.readFile('./input.txt', (err, data) => {
  if (err) throw err;
  console.log(data); // Buffer
  const output = data + '我又被寫出去了～';
  fs.writeFile('./output.txt', output, (err) => {
    if (err) throw err;
    console.log('It\'s saved!');
  });
});
```

Create Your Own Modules

- Use the **`exports`** keyword to make properties and methods available outside the module file.
  - `exports.myDateTime = function () { return Date(); };`
  - ```
    const math = require('math');
    exports.incrementOne = function (num) {
      return math.add(num, 1);
    };
    ```

- Save your module to a js file, say, “myModule.js”.
  To use this module in another file —
  - ```
    var http = require('http');
    var dt = require('./myModule');    // NO “.js”
    http.createServer(function (req, res) {
        res.writeHead(200, {'Content-Type': 'text/html'});
        res.write
          ("The date and time are currently: " + dt.myDateTime());
        res.end();
    }).listen(8080);
    ```

Node Package Manager (npm)

- NPM is a package manager for Node.js packages, or modules if you like.
  - https://www.npmjs.com hosts thousands of free packages to download and use.
  - The “npm” program is installed on your computer when you install Node.js
- A **package** in Node.js contains all the files you need for a module. **Modules** are JavaScript libraries you can include in your project.
- Let’s go through this tutorial: https://www.sitepoint.com/beginners-guide-node-package-manager/

Node Version Manager (nvm)

- 因為 node 的版本更新的速度太快了... 所以需要有一個版本管理工具，讓你可以無痕的在不同的 node 版本之間切換

- Let's go through this tutorial: https://www.keycdn.com/blog/node-version-manager/

After Class ◦ Week #5

- **Reading assignment**
  - A Beginner's Guide to HTTP and REST：https://code.tutsplus.com/tutorials/a-beginners-guide-to-http-and-rest--net-16340
  - REST API tutorial: http://www.restapitutorial.com/