**前端技术栈学习-基本操作、环境配置、常用命令等**

1. **来点水货**
2. **前端开发的历史和趋势**

<https://github.com/ruanyf/jstraining/blob/master/docs/history.md>

1. **什么是前端开发工程化？**

<https://blog.csdn.net/mayfla/article/details/78697020>

1. **Cmder（终端、命令行工具）**

**功能：**

windows终端，取代系统自带的cmd

**安装：**

去官网http://cmder.net/下载安装即可，分别有mini（6MB）和full（84MD）版本，都是portable的，解压即可使用。

**技巧：**

1、使用命令行直接调用cmder：将安装目录添加到“环境变量”系统/path里，如D:\Program Files\cmder，加完之后,Win+r一下输入cmder,即可。

2、将cmder加入右键菜单：在管理员权限的终端输入以下语句即可: Cmder.exe /REGISTER ALL

3、一些设置：

字体：Main->Main console font ：“Source Code Pro 24”；

颜色： Features->Colors->Schemes ： “<xterm>”

**命令：**

cd 切换目录，**可输入唯一文件名的前面几个字符按tab键自动补齐文件名**

**cd /切换到盘符根目录**

**cd .. 切换到上级目录**

dir 显示文件，dir/w 仅显示文件命名，**dir .显示当前目录，dir ..显示上级目录**

cls 清屏

ctrl+c终止当前批处理操作

mkdir flodName创建文件夹

touch index.js 创建 index.js

mv index.js app.js src/ 把index.js app.js移动到src目录下

rm build.js 删除build.js

rm -rf build 删除文件夹

1. **Hbuilder（前端IDE工具）**
2. **px自动转rem**

“工具”->“选项”->“Hbuilder”->“代码助手设置”->“px自动转rem设置”：比例按设计图尺寸/10

1. **主题/字体：**Source Code Pro 15 pt.
2. **语法预编译：**

.sass,.scss

C:\Ruby25-x64\bin\scss.bat

--no-cache %FileName% ../css/%FileBaseName%.css --style compact --precision 6

1. **Emmet**

Hbuilder安装目录\plugins\io.emmet.eclipse\_版本号.jar直接修改压缩包里的io\emmet\snippets.json自定义缩写名称连接符用“:”无效，用“-”,实际运用中wb:a和wb-a都可以生成

1. **Templates(自定义模板)**

C:\Users\Administrator\HBuilder settings

1. 11
2. 11
3. 1
4. **Git（版本控制工具）**

**敲重点：搞清楚集中式和分布式；搞清楚工作区和暂存区**



**功能：**

版本控制，代码分享

**技巧：**

1. 忽略文件/文件夹：项目更目录下使用Git命令“touch .gitignore”添加配置文件，输入需要忽略的文件/文件夹

**命令：**

**创建版本库：**

git init #初始化所在目录为Git仓库，初始化的目录可以不为空

**添加文件到版本库：**

git add <file> ... #添加文件到暂存区（stage）

-f <file> ... #强制添加到暂存区（可用于添加忽略文件）

git commit -m "提交说明" #从暂存区提交到版本库

git add 命令可多次执行，然后commit一次。

**时光穿梭机**

**查看状态、差异：**

git status #查看仓库当前状态

git diff [file] #比较工作区和暂存区的差异

git diff --cached [file] #比较暂存区和版本库的差异

git diff HEAD -- [file] #比较工作区和版本库的差异

**版本切换：**

git log #查看提交历史

git log -1 #查看最后一次提交信息（-2 则是最后两次）

git log --pretty=oneline #单行格式显示提交历史

--graph #显示分支合并图

--abbrev-commit #简写的commit\_id

git reflog #查看所有操作记录，包括删除的commit记录

git reset --hard HEAD^ #回退到上一版本

# HEAD 表当前版本， HEAD^ 表上一版本，HEAD^^ 表上两版本，HEAD~99 表上99版本。

git reset --hard commit\_id #切换到指定版本

Git跟踪管理的是修改，而非文件

**撤销修改：**

git checkout -- <file> #撤销工作区的修改

git reset HEAD <file> #撤销暂存区的修改

若已commit但没提交到远程库，可用版本回退进行撤销

**删除文件：**

方法一：工作区删除文件，然后正常提交

rm <file> ... #工作区删除

git add <file> ... #将修改提交至暂存区

git commit -m "说明" #提交到版本库

方法二：直接命令删除工作区和暂存区，然后提交版本库

git rm <file> ... #删除工作区和暂存区文件

git commit -m "说明" #提交到版本库

删除、增加文件也都属于修改

**远程仓库**

**创建SSH Key：**

ssh-keygen -t rsa "youremail@example.com" #生成的Key在C:\Users\xxx\.ssh文件夹里面，pub后缀是公钥，另一个是私钥。

**添加远程库：**

git remote add origin git@server-name:path/repo-name.git #添加远程仓库

git remote #查看远程库信息

-v #显示详细信息

git push -u origin <branch> #推送并关联指定分支到远程库

除第一次关联，之后push不用加-u选项

**从远程库克隆：**

git clone git@server-name:path/repo-name.git #将远程仓库克隆到当前目录

git pull #拉取远程仓库内容

**分支管理**

git branch <branch> #创建分支

git checkout <branch> #切换到指定分支

git checkout -b <branch> #创建并切换到该分支

git branch #查看现有分支

git branch -d <branch> #删除指定分支

git branch --set-upstream <branch\_local> <branch\_remote> #指定本地分支与远程分支的链接

git merge <branch> #合并指定分支到当前分支

--no-ff <branch> #禁用快速合并

git merge --no-ff -m "提交说明" <branch> #普通方式合并，并附提交说明

git stash #保存当前工作环境（包括工作区和暂存区）

git stash list #查看保存的工作列表

git stash apply [stash@{X}] #恢复工作状态，但不删除stash内容

git stash pop [stash@{X}] #恢复工作状态，并删除stash内容

git stash drop [stash@{X}] #删除stash内容

git branch -D <branch> #强制删除分支（常用于未合并的分支）

HEAD不是直接指向提交点，而是指向分支，分支再指向提交点

**多人协作**

##error: failed to push some refs to ...

1. git pull 远程库

2. 解决冲突（若有），再push

**标签管理：**

git tag #查看现有标签

git tag <tag\_name> #给当前所在的commit打标签

git tag <tag\_name> <commit\_id> #给指定commit打标签

git tag -a <tag\_name> -m "标签说明" <commit\_id> #给指定commit打标签，并附说明

-s <tag\_name> -m "标签说明" <commit\_id> #用gpg私钥签名

-d <tag\_name> #删除标签

git show <tag\_name> #显示标签信息

git push origin <tag\_name> #推送标签到远程库

git push origin --tags #推送所有未推送的标签到远程库

git push origin :refs/tags/<tag\_name> #删除远程标签（先删除本地，再使用该命令删除）

**自定义Git：**

git config --global user.name "you\_name" #设置全局用户名

git config --global user.email "email@example.com" #设置全局邮箱

git config --global color.ui true #设置全局颜色显示

git config --global alias.<alias\_name> <'command\_name'> #设置别名

**忽略特殊文件：**

工作区创建.gitignore文件

内容举例，如下：

#Windows:

Thumbs.db

ehthumbs.db

Desktop.ini

#Python:

\*.py[cod]

\*.so

\*.egg

\*.egg-info

dist

build

#My configurations:

db.ini

deploy\_key\_rsa

git check-ignore -v <file> #查看忽略该文件的规则

**配置别名列表：**

git config --global alias.confg 'config --global'

git confg alias.st status

1. **Nodejs（js运行环境/平台）**

<https://nodejs.org/en/>

1. **Npm（包管理器）**

**功能：**

随同NodeJS一起安装的**包管理工具**，常见使用场景：从NPM服务器下载第三方包到本地或命令行程序到本地使用；上传自己的包或命令行程序到NPM服务器供别人使用。

**技巧：**

1. **-save-dev 可简写成 -D**
2. **npm install \*安装包的时候太卡，可使用淘宝镜像cnpm install \***

npm install -g cnpm --registry=https://registry.npm.taobao.org

1. **安装包卡在 node scripts/install.js 解决办法**

先在终端运行 npm config set registry https://registry.npm.taobao.org

然后编辑npm目录下的~/.npmrc文件，加入下面内容

registry=https://registry.npm.taobao.org

sass\_binary\_site=https://npm.taobao.org/mirrors/node-sass/

phantomjs\_cdnurl=http://npm.taobao.org/mirrors/phantomjs

ELECTRON\_MIRROR=http://npm.taobao.org/mirrors/electron/

**命令：**

npm install <name> 安装nodejs的依赖包

npm install <name> -g 将包安装到全局环境中

npm install <name> --save 安装的同时，将信息写入package.json中

npm init 会引导你创建一个package.json文件，包括名称、版本、作者这些信息等

Npm init --yes 初始化package.json文件，自动输入信息

npm remove <name>移除

npm update <name>更新

npm ls 列出当前安装的了所有包

npm root 查看当前包的安装路径

npm root -g 查看全局的包的安装路径

npm help 帮助，如果要单独查看install命令的帮助，可以使用的npm help install

npm -v 查询npm版本

1. **Gulp（前端工作流自动构建工具）**

**功能：**

Gulp 是一个基于 Node.js 的开源前端工作流构建工具。实现 Web 前端自动化开发的工具，利用它能够极大的提高开发效率。

**技巧：**

1. **.watch方法路径不要用 './xx'：** 用 './xx' 开头作为当前路径开始，会导致无法监测到新增、删除文件
2. **执行“默认组合任务”：** 当前案例为同时执行“auto（监控文件自动编译）”和“browser-sync（自动刷新浏览器）”两项任务，每次改变了js、css、html就自动编译、压缩文件，并刷新浏览器
3. **发布项目：** 把依赖库node\_modules删除；**提交到git：** 使用.gitignore忽略规则添加node\_modules
4. **拿到新项目：**先npm/cnpm install安装所有依赖库

**匹配符 \*、\*\*、！、{}**

gulp.src('./js/\*.js') // \* 匹配js文件夹下所有.js格式的文件

gulp.src('./js/\*\*/\*.js') // \*\* 匹配js文件夹的0个或多个子文件夹

gulp.src(['./js/\*.js','!./js/index.js']) // ! 匹配除了index.js之外的所有js文件

gulp.src('./js/\*\*/{omui,common}.js') // {} 匹配{}里的文件名

**Gulp 能够解决哪些问题：**

1. 文件清理 （del gulp-clean）
2. 文件重命名 （gulp-rename）
3. 文件拷贝 （gulp-copy）
4. 文件合并 （gulp-concat）
5. 文件压缩 （gulp-uglify gulp-minify）
6. 文件过滤 （gulp-filter）
7. css相关

css压缩 （gulp-csso gulp-clean-css）

css 自动添加前缀 （gulp-autoprefixer）

sass/scss，less转换 （gulp-sass gulp-less）

1. js相关

es6转换成es5 （gulp-babel）

1. html相关

html压缩 （gulp-minify-html gulp-htmlmin）

jade转换成html （gulp-jade）

1. 图片相关

图片压缩 （gulp-imagemin）

制作雪碧图 （gulp.spritesmith gulp-css-sprite）

Svg转换成png （gulp-svg2png）

1. 其他

从依赖库package.json里加载gulp插件 （gulp-load-plugins）

自动刷新浏览器 （browser-sync）

按顺序同步执行任务 （run-sequence gulp-sequence）

**实操：**

**1、安装gulp和插件:**

全局安装：npm install gulp -g

当前项目安装：npm install gulp --save-dev

初始化项目package.json的配置：npm init --yes

安装压缩js的插件：npm install gulp-uglify --save-dev

**2、项目根目录gulpfile.js中，输入任务处理代码**

var gulp = require( 'gulp' );

var uglify = require( 'gulp-uglify' );

gulp.task('min-js', function() {

gulp.src('src/js/\*.js')

.pipe( uglify() )

.pipe( gulp.dest('dist/js') );

});

gulp.task(name, fn) - 定义任务，第一个参数是任务名，第二个参数是任务内容。

gulp.src(path) - 选择文件，传入参数是文件路径。

gulp.dest(path) - 输出文件

gulp.pipe() - 管道，你可以暂时将 pipe 理解为将操作加入执行队列

另一种方法：使用gulp-load-plugins加载插件

var $ = require('gulp-load-plugins')(); // $ 是一个对象,加载了依赖里的插件

gulp.src('./\*\*/\*.js')

.pipe($.concat('all.js')) // 使用插件就可以用$.PluginsName()

.pipe($.uglify())

.pipe($.rename('all.min.js'))

.pipe(gulp.dest('./dist'))

**3、执行任务**

在命令行/终端下执行任务: gulp min-js

**4、常用插件请参照**

<https://segmentfault.com/a/1190000008349859>

<https://blog.csdn.net/qq_27626333/article/details/78006271>

1. **Webpack（模块打包工具）**

**敲重点：搞清楚开发环境和生产环境**

**功能：**

WebPack可以看做是模块打包机：它做的事情是，分析你的项目结构，找到JavaScript模块以及其它的一些浏览器不能直接运行的拓展语言（Scss，TypeScript等），并将其转换和打包为合适的格式供浏览器使用。

**三个核心思想**

1. **万物皆模块：**在 webpack 的世界中，任何资源都可以当做模块的方式引用；
2. **按需加载：**当应用体积增大，实现代码的按需加载是刚需，这也是 webpack 出现的根本原因，特别是首屏加载时间要尽量减少；
3. **可定制化：** 任何一个工具都不可能解决所有问题，提供解决方案才是最可行的，webpack 基于可定制化的理念构建，通过插件系统，配置文件，可以实现大型项目的定制需求。

**技巧&踩坑：**

1. **webpack-dev-server**

这是一个小型的Node.js Express服务器，能实现类似于gulp的自动刷新功能。

**最需要记住的一点：webpack-dev-server生成的文件并没有放在你的真实目录中,而是放在内存中，webpack.config.js配置中的output.publicPath即该虚拟路径。**

在终端/以命令行启动webpack-dev-server时，期望**自动刷新**，需要注意三点：

1）、在命令行中添加--inline，webpack-dev-server --inline

2）、在webpack.config.js中添加devServer:{inline:true}

3）、命令行不要添加--hot，webpack.config.js同样不要加hot，因为热替换页面是不会刷新的

如果期望**监控文件改动并自动刷新**，需要在命令行中添加--watch

**自动打开+监控文件+自动刷新**

**webpack-dev-server --open --watch --inline**

1. **Package.json配置中，除start可以用npm start运行，其他npm run xxx**
2. 启用Source maps使调试更容易，可以跟踪定位到你的开发代码，devtool: "eval-source-map"，有安全隐患，建议只在开发模下使用
3. **11**
4. **111**
5. **Vue**