**前端技术栈学习-基本操作、环境配置、常用命令等**

1. **Cmder**

**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**

dir 显示文件，dir/w 仅显示文件命名

cls 清屏

1. **Nodejs**
2. **Npm**
3. **Gulp**

**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. 发布项目把依赖库node\_modules删除
2. 拿到别人的项目，先npm/cnpm install
3. Ctrl+c终止当前批处理操作

**Gulp 能够解决哪些问题：**

1. 文件清理 (gulp-clean)
2. 文件拷贝 (gulp-copy)
3. 文件转换 （gulp-webpack）
4. 文件合并 （gulp-concat)
5. 文件压缩 （gulp-uglify gulp-minify)
6. 文件服务 （gulp-connect)
7. 文件监控 （gulp-watch)
8. css 相关

less，sass 转换 (gulp-less ，gulp-sass)

css 自动添加前缀 (gulp-autoprefixer)

1. js 相关

jslint (gulo-eslint)

10、html 转换

html 模板 (gulp-jade，gulp-ejs)

html prettify

html validator

html minifier

**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

**3、执行任务**

在命令行/终端下执行任务: gulp min-js

1. **Webpack**
2. **Git、**

**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 -C "youremail@example.com" #生成的Key在家目录.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. **Vue**