# svn

[javascript\SVN\SVN.html](file:///C:\Users\Administrator\AppData\Roaming\Microsoft\Word\javascript\SVN\SVN.html)

# Git

安装与配置

下载 地址：<https://git-for-windows.github.io>

配置个人信息（名字与邮箱） ，使用Git的第一件事就是设置你的名字和email,这些就是你在提交commit时的签名

git config --global user.name "Your Name"

git config --global user.email "email@example.com"

查看是否配置成功，用命名git config -l

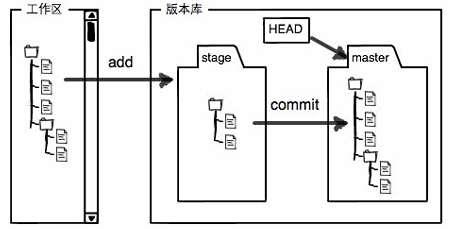
git的使用（命令行）

创建本地仓库

**git init** 把当前目录变成一个git仓库，并自动创建master分支

以上命令会在当前目录下创建了一个.git 隐藏目录，它就是所谓的Git 仓库。生成仓库后的目录就不是普通的文档目录了，我们将其称为工作区，所以工作区中都包含一个git仓库，而一个git仓库中又包含一个暂存区和一个版本库

工作区与版本库概念



Git有三大区：工作区、暂存区、版本库

工作区（Working Directory）：电脑中的目录

仓库：工作区有一个隐藏目录.git，这个不算工作区，而是Git仓库。

暂存区（stage/index）：一个临时的存储区域

版本库 （Repository）

添加文件到版本库的步骤（重点）

1创建（修改）文件 ，往工作区中添加/修改文件

2添加到暂存区：**git add** <file> ，把修改存放到暂存区

git add 文件夹：把文件夹下的所有修改添加到暂存区

git add **.** ：添加所有修改到暂存区

3提交到版本库：**git commit -m "**备注**"**

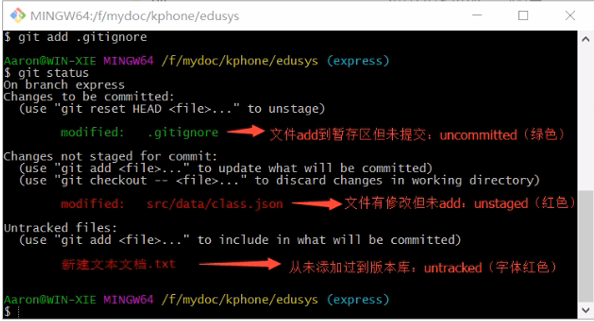
使用git commit 命令可将暂存区的内容提交至版本库中，这个过程称为提交，每一次提交都意味着版本在进行一次更新（会自动生成一个commit id）

PS：如果不写-m回车会进入vim编辑界面，退出方法：输入:q => 回车

其他辅助命令

**git status** 查看仓库变更状态

用status查看仓库会有几种状态：untracked、unstaged、uncommitted



git远程仓库

注册登录

github(国外): https://github.com/

coding(国内):https://coding.net

关联本地仓库与远程仓库

有两种方式关联：ssh和https两种协议，https比较简单，但提交时每次都分输入用户名和密码，如使用https协议，直接跳过以下第1、第2步

1创建SSH Key：**ssh-keygen -t rsa -C 'email地址'**

以上命令会在当前window用户的目录里创建.ssh目录，里面有id\_rsa（私钥）和id\_rsa.pub（公钥）两个文件

2添加SSH Key到Git服务器

添加完成后测试线路是否连通：**ssh -T git@github.com**

3建立本地仓库与远程仓库的连接

方式1：适用于先有本地仓库，后有远程仓库的情况

格式：git remote add 远程仓库名 远程仓库地址

**git remote add origin** git@github.com:xxx/view.git

PS：删除远程仓库连接：git remote remove 远程仓库名

方式2：克隆（适用于先有远程库，后有本地仓库的情况）

格式：**git clone** 远程仓库地址

当你从远程仓库克隆时，实际上Git自动把本地的master分支和远程的master分支对应起来了，并且，远程仓库的默认名称是origin

推送到远程仓库（重点）

**git push**

格式：git push 远程仓库名 本地分支名:远程分支名

把本地分支内容推送到远程分支（远程分支名省略表示推送到与本地分支相同的分支）

git push origin master

拉取与合并（重点）

**git pull**

格式：git pull 远程仓库名 远程分支名:本地分支名

拉取远程分支内容到本地并与本地分支进行合并（本地分支名省略表示合并到与远程分支名相同的分支）

git pull origin master

git fetch 拉取远程分支内容

git merge 合并分支内容

git pull origin master

//以上命令相当与以下命令等效

git fetch origin master

git merge origin/master

\*\*push和pull后的分支顺序格式：<来源地>:<目的地>

版本回退 git reset

git reset --hard HEAD^ 回退到上一个版本

git reset --hard [commit id] 回退到指定版本（版本号没必要写全，前几位就可以了，Git会自动去找。）

git reset --hard [commit id] <file> 回退指定文件

参数说明 –hard:工作区、暂存区、版本库的文件同时回退

–mixed：暂存区、版本库的文件回退（默认）

–soft：仅仅回退版本库中的文件

当前版本：HEAD

上一个版本：HEAD^

上上个版本：HEAD^^ … 依此类推

前100个版本：HEAD~100

**git log** 显示从最近到最远的提交日志

–pretty=oneline（显示简要信息id+备注）

–graph（图形显示版本走向）

–abbrev-commit（显示简写的id）

一大串类似3628164…882e1e0的是commit id（版本号）

git reflog 查看命令历史

git checkout -- <file> 放弃工作区的修改

git rm --cache <file> 撤销暂存区的修改

git reset HEAD <file> 撤销暂存区的修改

git diff <file> 对比文件

git过滤配置(过滤清单)

一般来说每个Git项目中都需要一个“**.gitignore**”文件，这个文件的作用就是告诉Git哪些文件不需要添加到版本管理中。

#过滤dist根目录下的文件（不过滤其他目录下的dist文件夹）

/dist

#过滤所有mtk文件夹

mtk/

#过滤所有.zip文件

\*.zip

#过滤某个具体文件

/mtk/do.c

#为注释，被过滤掉的文件就不会出现在你的GitHub库中了，当然本地库中还有，只是push的时候不会上传。

分支操作（了解）

创建分支 git branch 分支名

切换分支 git checkout 分支名

以上两步合并为：git checkout -b 分支名

查看分支 git branch

列出所有分支，当前分支前面会标一个\*号

合并分支 git merge 分支名

git merge dev：把dev分支合并到当前分支

Fast-forward：快速合并，禁用快速合并： –no-ff （保持分支信息）

合并要创建一个新的commit，所以加上-m参数，把commit描述写进去

删除分支 git branch -d 分支名

强行删除，需要使用命令git branch -D feature-vulcan

获取远程分支

先获取（git fetch），然后在本地创建一个同名分支,并将远程分支映射到此分支（git branch dev origin/dev）

在实际开发中，我们应该按照几个基本原则进行分支管理：

首先，master分支应该是非常稳定的，也就是仅用来发布新版本，平时不能在上面干活；

那在哪干活呢？干活都在dev分支上，也就是说，dev分支是不稳定的，到某个时候，比如1.0版本发布时，再把dev分支合并到master上，在master分支发布1.0版本；

你和你的小伙伴们每个人都在dev分支上干活，每个人都有自己的分支，时不时地往dev分支上合并就可以了。

# 模块化开发

requirejs

RequireJS的目标是鼓励代码的模块化，它使用在脚本中加载js的方式代替传统的script加载步骤。可以用它来加速、优化代码，但其主要目的还是为了代码的模块化

官网：<http://requirejs.org/>  
中文网：<http://www.requirejs.cn/>

使用

按照requireJS当中的规范要求，只需加载一个文件就可以了，但同时需要带上一个配置文件config.js

<script data-main ="config.js" src ="require.js"></script>

配置config

这个require.js可以从官网下载，但config.js必须自己完成，写起来非常简单:

//config.js-----

**requirejs.config({**

baseUrl : "js",

paths : {

"jquery": "lib/jquery.1.11.3"

}

**})**;

**baseUrl**的意思就是说你所有的路径都是基于它基础上的

如未显式设置baseUrl，则默认值是加载require.js的HTML文件所在的路径

但如果用了data-main属性，则该路径就变成baseUrl

**paths**映射那些不直接放置于baseUrl下的模块名。设置path时起始位置是相对于baseUrl

冒号左边jquery模块名称，右边则是文件路径，注意文件没有后缀名，因为在require看来所有的模块都是JS文件

**shim**依赖(了解)有一些插件，本身return出任何对象或函数，只是对某个框架的扩展，例如 jquery.scroll.js 该怎么实现模块化加载，答案是shim配置

requirejs.config({

baseUrl: "js",

paths: {

"jquery" : "lib/jquery.1.11.3",

"jquery.scroll" : "lib/jquery.scroll"

},

shim: {

"jquery.scroll": {

deps: ["jquery"],

exports: "jQuery.fn.scroll"

}

});

常用方法

配置参数：requirejs.config/require.config

加载模块：requirejs/require

定义模块：define

加载模块

以上配置好后，就可以加载模块了，如加载jquery只需如下代码

requirejs(['jquery'],function(jq){

//这里得到jquery

});

定义模块define

使用defined方法定义模块必须符合require的规范要求，格式如下：

//把commonjs定义成模块

define(function(){

return {

getStyle : function(){ ........ },

randomColor : function(){ ......... }

}

});

如果定义的模块需依赖其他模块，格式如下，这种模块加载方式称为：**预加载**

define(["jquery"],function(jq){

//根据设定，模块需要使用jquery

//define函数中的第一个参数，用来表明，我们即将引用哪个模块。由于之前我们已经定义过了jquery模块，所以这里可以直接使用

//注意，你如果不写声明，这里是用不了jQuery的

//你如果注意到第一个参数是一个数组，那就应该猜到，其实我们可以一次引用好多模块

//jq变量就是jquery，你如果不习惯，把名字改回$就是了

return {

start: function(){

jq("#box").show(1000);

console.log("模块提供的start方法");

//为什么这里要return呢？ 因为我们定义的模块要被其他人使用

//那又为什么要return对象呢？为了更符合模块化思维以及面相对象的原则

//我们返回一个对象而不是函数，使用会更方便（待会讲完怎么调用你就知道了）

}

});

如果需要在define内部依赖其他模块，则写法如下，这种模块加载方式称为：**延迟加载**

define(["require","jquery"],function(require){

var a = require('./js/a');

});

模块化开发的优点

避免命名冲突：整个项目当中，再也没有出现一个全局变量，即使两个框架名字冲突了也没关系

更好的依赖处理：不用再考虑过加载顺序的问题

按需加载：更好的代码组织方式，所有的JS文件的加载过程，已经变成了异步

面向对象编程：用了requireJS，你的代码想不写成面相对象都难

模块化规范与框架

1.commonJS：通用的模块规范（同步）；  
模块化框架：nodejs

2.AMD：异步模块定义规范（预加载）；  
模块化框架：requirejs

3.CMD：通用模块定义规范（延迟加载）。  
模块化框架：seajs  
官网：<http://seajs.org>  
五分钟上手教程：<http://seajs.org/docs/#quick-start>

# gulp

gulp是当下最流行的自动化工具 ，可以自动化完成我们开发过程中大量的重复工作。

为什么要用自动化构建工具？ 主要是用来自动完成一系列重复的操作，如：

编译：less->css

sass->css

coffeescript->js

es6->es5

合并: css, js

压缩 ：css, js, html

优化：图片优化

官网：<http://gulpjs.com/>  
中文网：<http://www.gulpjs.com.cn/>

安装与运行

前提是安装了nodejs环境

1全局安装 gulp：

**npm install --global gulp**

说明：全局安装gulp目的是为了通过它执行gulp任务

2本地安装gulp：

作为项目的开发依赖（devDependencies）安装：

**npm install --save-dev gulp**

本地安装gulp是为了调用gulp插件的功能，

– –save-dev 保存配置信息至 package.json 的 devDependencies 节点

这步操作前先新建package.json文件（npm init）

这步完成后就可以安装各种gulp插件了

3创建**gulpfile.js**文件

在项目根目录下创建一个名为 gulpfile.js 的文件（重要）

说明：gulpfile.js是gulp项目的配置文件，是位于项目根目录的普通js文件，内容如下：

var gulp = require('gulp');

gulp.task('default', function() {

// 将你的默认的任务代码放在这

});

4运行 gulp：

在命令行执行以下命名，如果不写任务名称，则自动运行default任务（如果有）

**gulp <任务名称>**

gulp工作流程

1. 选通过 gulp.src() 方法获取到想要处理的文件，并返回文件流
2. 然后文件流通过 pipe 方法导入到 gulp 的插件中
3. 经过插件处理后的文件流再通过pipe方法导入到 gulp.dest() 方法中
4. 最后通过gulp.dest() 方法把流中的内容写入到文件中

PS：文件流=>文件在内存中的状态

gulp API

创建任务

gulp.task('buildsass',function(){

});

匹配要处理的文件

gulp.src(globs[, options])

globs匹配语法：https://github.com/isaacs/node-glob（底部PS）

options 有3个属性buffer、read、base

输出文件

gulp.dest(path[, options]) 把文件流中的内容中输出到指定目录

监听文件修改，并执行相应任务

gulp.watch(glob [, opts], tasks)

gulp.watch(glob [, opts, cb])

gulp插件

[project\doc\gulp常用插件.html](project/doc/gulp常用插件.html)

大部分插件都可以在http://[www.npmjs.com](http://www.npmjs.com)找到，任何插件的使用都要经历以下三步：

1安装插件：npm

npm install --save gulp-htmlmin

PS：可一次性安装多个插件，插件间用空格隔开

2引包：require()

var htmlmin = require('gulp-htmlmin')

3使用：pipe()

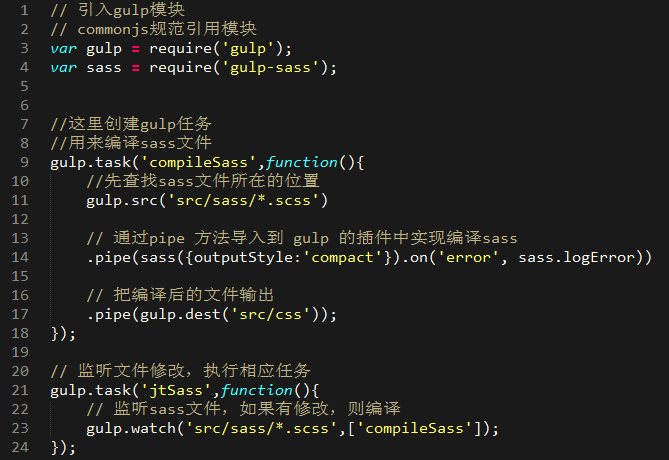
gulp.task('htmlmin',function(){

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

.pipe(htmlmin())

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

});



常用gulp插件

* htmml压缩：gulp-htmlmin
* css压缩：gulp-clean-css
* js压缩：gulp-uglify
* 合并文件：gulp-concat
* 文件重命名：gulp-rename
* 编译Sass: gulp-sass
* 编译 Less：gulp-less

其他常用插件

* 浏览器同步测试：browser-sync
* 创建node服务器：http-server

globs语法

globs需要处理的源文件匹配符路径，语法如下

匹配单个文件： gulp.src('src/js/index.js')

匹配多个文件： gulp.src(['src/js/index.js','src/js/detail.js']) //多个文件以数组形式传入

匹配所有文件： gulp.src('src/js/\*.js')

匹配符： !：排除文件，

\*：匹配所有文件，

\*\*：匹配0个或多个子文件夹，

{}：匹配多个属性



# Sass

语法

注释

sass有两种注释方式（与js相同）多行注释与单行注释

body {

color: #333; // 这种注释内容不会出现在生成的css文件中

padding: 0; /\* 这种注释内容会出现在生成的css文件中 \*/

}

变量

sass的变量必须是**$开头**，**后面紧跟变量名**，而变量值和变量名之间就需要使用冒号**:分隔**开（就像CSS属性设置一样）。

全局变量与局部变量：定义在任何选择器之外的变量被认为是全局变量，定义在选择器内的变量称之为局部变量。

但启用了**!global**后，即使写在局部也能覆盖**全局**变量（sass 3.4版本后可用

$color:#fff !global;

默认变量：sass的默认变量仅需要在值后面加上!default即可。

$fontSize:12 !default;

覆盖的方式也很简单，只需要在默认变量之前重新声明下变量即可:

$fontSize:16;

$fontSize:12 !default;

特殊变量：一般我们定义的变量都为属性值，可直接使用，但是如果变量作为属性或在某些特殊情况下等则必须要以**#{$**variables**}**形式使用

$borderDirection:top !default;

//应用于class和属性

.border-#{$borderDirection}{

border-#{$borderDirection}:1px solid #ccc;

}

多值变量：多值变量分为list类型和map类型，简单来说list类型有点像js中的数组，而map类型有点像js中的对象。

//list类型

$pd: 5px 10px 20px 30px;

//使用

.content{padding:$pd;}

//map类型

$headings: (h1: 2em, h2: 1.5em, h3: 1.2em);

//使用

h1{map-get($headings,h1)}

嵌套(Nesting)

#content {

article {

h1 { color: #333 }

p { margin-bottom: 1.4em }

}

aside { background-color: #EEE }

}

在嵌套中用**&**表示父元素选择器

混合器(mixin)

变量可以实现简单样式的重用（如color,width等），但是当你的样式变得越来越复杂，你需要大段大段的重用样式的代码，可以通过sass的混合器实现重用

sass中使用**@mixin**声明混合，通过**@include**来调用

无参数mixin

有参数mixin：参数名以$符号开始

多个参数mixin：多个参数以逗号分开

@content

@mixin max-screen($res){

@media only screen and ( max-width: $res )

{

@content;

}

}

@include max-screen(480px) {

body { color: red }

}

PS：@mixin通过@include调用后解析出来的样式是以拷贝形式存在的，而下面的继承则是以联合声明的方式存在的，所以从3.2.0版本以后，建议传递参数的用@mixin，而非传递参数类的使用下面的继承%。

继承 @extend

继承一般样式 @extend h1

占位选择器%

//占位符编译后不存在css样式中

%ir{

color: transparent;

text-shadow: none;

background-color: transparent;

border: 0;

}

@extend %ir;

函数

Sass中的数字函数提要针对数字方面提供一系列的函数功能：

常用函数：

percentage($value)：将一个不带单位的数转换成百分比值；

round($value)：将数值四舍五入，转换成一个最接近的整数；

ceil($value)：将大于自己的小数转换成下一位整数；

floor($value)：将一个数去除他的小数部分；

abs($value)：返回一个数的绝对值；

min($numbers…)：找出几个数值之间的最小值；

max($numbers…)：找出几个数值之间的最大值。

lighten($color,$percent)

darken($color,$percent)，$color颜色值，$percent百分比

自定义函数 格式：@fuction 函数名

$oneWidth: 10px;

$twoWidth: 40px;

@function widthFn($n) {

@return $n \* $twoWidth + ($n - 1) \* $oneWidth;

}

.leng {

width: widthFn($n : 5);

}

运算

sass具有运算的特性，可以对数值型的Value(如：数字、颜色、变量等)进行加减乘除四则运算。请注意运算符前后请留一个空格，不然会出错。

条件判断及循环

@if判断 可一个条件单独使用，也可以和@else结合多条件使用

@if $type == ocean {

color: blue;

} @else if $type == matador {

color: red;

} @else {

color: black;

}

@for循环

@for $var from <start> through <end>（包含end值）

@for $var from <start> to <end>（不包含en值）

导入

sass中导入其他sass文件，最后编译为一个css文件，优于纯css的@import

@import 'reset';

编译

**gulp-sass**

参数outputStyle：nested(默认）expanded展开compact单行 compressed压缩

gulp.task('sass', function () {

return gulp.src('./sass/\*\*/\*.scss')

.pipe(sass({outputStyle: 'compressed'}).on('error', sass.logError))

.pipe(gulp.dest('./css'));

});

//文件监听（文件有修改自动编译）

gulp.task('sassWatch',function(){

gulp.watch('./src/sass/\*.scss',['sass']);

});

文件名以\_开头的sass文件不会被编译成css文件