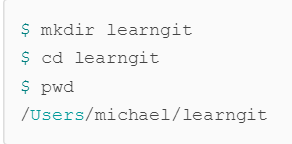
Git：版本控制系统，能允许别人编辑，记录每次文件的改动

局域网是在一个局部的地理范围内将各种计算机，外部设备和数据库等互相联接起来组成的计算机通信网。

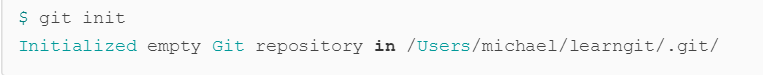
集中式版本控制系统，必须联网才能工作，版本库是集中存放在中央服务器的，

**创建仓库：**

首先，选择一个合适的地方，创建一个空目录：



第二步，通过git init命令把这个目录变成Git可以管理的仓库：



如果你没有看到.git目录，那是因为这个目录默认是隐藏的，用ls -ah命令就可以看见。

版本控制系统没法跟踪Word文件的改动的，要以纯文本方式编写文件。建议你下载[Notepad++](http://notepad-plus-plus.org/" \t "_blank)代替记事本，

**把一个文件放到Git仓库只需要两步。**

1. 使用命令git add <file>，注意，可反复多次使用，添加多个文件；
2. 使用命令git commit -m <message>，完成。

要随时掌握工作区的状态，使用git status命令。

如果git status告诉你有文件被修改过，用git diff可以查看修改内容。

**版本回退**

HEAD指向的版本就是当前版本，因此，Git允许我们在版本的历史之间穿梭，使用命令git reset --hard commit\_id。

穿梭前，用git log（git log命令显示从最近到最远的提交日志）可以查看提交历史，以便确定要回退到哪个版本。

要重返未来，用git reflog查看命令历史，以便确定要回到未来的哪个版本。

需要提交的文件修改通通放到暂存区，然后，一次性提交暂存区的所有修改。

为什么Git比其他版本控制系统设计得优秀，因为Git跟踪并管理的是修改，而非文件。

**撤销修改**

场景1：当你改乱了工作区某个文件的内容，想直接丢弃工作区的修改时，用命令git checkout -- file。

场景2：当你不但改乱了工作区某个文件的内容，还添加到了暂存区时，想丢弃修改，分两步，第一步用命令git reset HEAD <file>，就回到了场景1，第二步按场景1操作。

场景3：已经提交了不合适的修改到版本库时，想要撤销本次提交，参考[版本回退](https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/0013744142037508cf42e51debf49668810645e02887691000" \t "_blank)一节，不过前提是没有推送到远程库。

:q 退出 git add \* 添加目录里全部文件到暂存区

，用git diff HEAD -- readme.txt命令可以查看工作区和版本库里面最新版本的区别

git commit -m <message>， git reset --hard commit\_id。

git checkout -- file。 git diff HEAD -- readme.txt

**删除文件**

直接在文件管理器中把没用的文件删了，或者用rm命令删除

（1）确实要从版本库中删除该文件，那就用命令git rm删掉，并且git commit。

（2）如果误删，用git checkout撤销删除

**远程仓库(**GitHub上的仓库既可以作为备份，又可以让其他人通过该仓库来协作**)**

第1步：创建SSH Key

打开Shell（Windows下打开Git Bash）

$ ssh-keygen -t rsa -C [youremail@example.com](mailto:youremail@example.com)

【cd ~(进入用户主目录)

cd .ssh(进入.ssh目录)

ls(列出.ssh目录的文件，可以看到id\_rsa id\_rsa.pub两个文件)

cat ~/.ssh/id\_rsa.pub(可以看到id\_rsa.pub文件的内容，复制即可)】

第2步：登陆GitHub，打开“Account settings”，“SSH Keys”页面：然后，点“Add SSH Key”，填上任意Title，在Key文本框里粘贴id\_rsa.pub文件的内容：

* $ git remote -v

**添加Ô¶³Ì²Ö¿â**

首先，登陆GitHub，然后，在右上角找到“Create a new repo”按钮，创建一个新的仓库：

在Repository name填入仓库名（origin），其他保持默认设置，点击“Create repository”按钮，就成功地创建了一个新的Git仓库：

在本地的learngit仓库下运行命令：

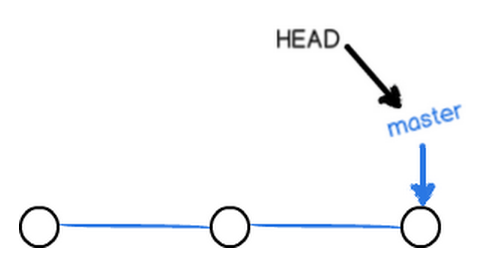
$ git remote add origin [git@github.com:michaelliao/learngit.git(账户名/](mailto:git@github.com:michaelliao/learngit.git(账户名/)仓库名.git) 来关联这个远程库

关联后，使用命令git push -u origin master第一次推送master分支的所有内容；

此后，每次本地提交后，只要有必要，就可以使用命令git push origin master推送最新修改；

第一次推送master分支时，加上了-u参数，Git不但会把本地的master分支内容推送的远程新的master分支，还会把本地的master分支和远程的master分支关联起来

**创建于合并分支**

HEAD严格来说不是指向提交，而才是指向提交的，所以，HEAD指向的就是当前分支，Git创建一个分支，即增加一个指针，切换分支实际上是改变HEAD的指向，使HEAD指向新的指针， 

查看分支：git branch

删除分支：git branch -d <name>（git branch -D <name>强行删除）

创建分支：git branch <name>

切换分支：git checkout <name>

创建+切换分支：git checkout -b <name>

合并某分支到当前分支：git merge <name>

当Git无法自动合并分支时，就必须首先解决冲突。解决冲突后，再提交，合并完成。

解决冲突就是把Git合并失败的文件手动编辑为我们希望的内容，再提交。

用git log --graph命令可以看到分支合并图。

合并分支时，加上--no-ff参数就可以用普通模式合并，合并后的历史有分支，能看出来曾经做过合并，而fast forward合并就看不出来曾经做过合并

**修复bug时**，我们会通过创建新的bug分支进行修复，然后合并，最后删除；

当手头工作没有完成时，先把工作现场git stash一下，然后去修复bug，修复后，再git stash pop，回到工作现场，你可以多次stash，恢复的时候，先用git stash list查看，然后恢复指定的stash，用命令：$ git stash apply stash@{0}

**多人协作**

可以在另一台电脑（注意要把SSH Key添加到GitHub）或者同一台电脑的另一个目录下克隆

你的小伙伴要在dev分支上开发，就必须创建远程origin的dev分支到本地，于是他用这个命令创建本地dev分支：

git pull失败了，原因是没有指定本地dev分支与远程origin/dev分支的链接，根据提示，设置dev和origin/dev的链接：

小结：

$ git branch --set-upstream-to=origin/dev dev

$ git checkout -b dev origin/dev

查看远程库信息，使用git remote -v；

本地新建的分支如果不推送到远程，对其他人就是不可见的；

从本地推送分支，使用git push origin branch-name，如果推送失败，先用git pull抓取远程的新提交；

在本地创建和远程分支对应的分支，使用git checkout -b branch-name origin/branch-name，本地和远程分支的名称最好一致；

建立本地分支和远程分支的关联，使用git branch --set-upstream branch-name origin/branch-name；

从远程抓取分支，使用git pull，如果有冲突，要先处理冲突。

rebase操作可以把本地未push的分叉提交历史整理成直线；

rebase的目的是使得我们在查看历史提交的变化时更容易，因为分叉的提交需要三方对比。

Js

Js需要由浏览器进行解释和执行

DOM是一套对 文档的内容进行抽象和概念化的方法

利用DHML把王爷标记为各种元素

利用CSS设置元素样式和它们的显示位置

利用JS实时操控页面和改变样式

JS允许直接对变量进行赋值而无需事先声明

在JS里用反斜线对字符进行转义：var mood=”don\’t ask”

Var bet=[ “john”, 1940,false]

用对象来代替传统数组，意味着可以用元素的名字，而不是下标数字来引用他们

Var lennon={name:”john”,year:1940,living:false}

+=：一次完成加法和赋值

=：赋值

==：只比较变量的值

===：既比较变量的值也比较变量的类型

**条件语句：**

在if语句里包含在花括号里的代码块只执行一次

If(条件)

{语句}

If(条件)

{语句}

else

{语句}

如果希望那些包含在循环语句内部的代码至少执行一次，用do…while循环

do

{语句}

While(条件)

For循环是while循环的一种变体，它的循环控制结构更加清晰，它的常见用途是对某个数组里的全部元素进行遍历处理

**函数**

把传递给函数的数据称为参数

命名变量时用下划线来分隔各个单词（var my\_name），在命名函数时，从第二个单词开始，把每个单词的第一个字母大写(var myName)

函数以外声明的变量为全局变量，函数内部var声明的为局部变量，若没有关键字var则视为全局变量**函数**

把传递给函数的数据称为参数

命名变量时用下划线来分隔各个单词（var my\_name），在命名函数时，从第二个单词开始，把每个单词的第一个字母大写(var myName)

函数以外声明的变量为全局变量，函数内部var声明的为局部变量，若没有关键字var则视为全局变量

对象是一个集合，比如人，而你就是这个对象里的一个实例，你有自己的属性和方法。

对象有自己的属性和方法，可以利用他的属性和方法来检索有关于它的信息

3种获取特定元素的方法：

getElementById, getElementByTagName, getElementByClassName,

getAttribute(获取属性) 与 setAttribute（修改属性）方法只能用于元素节点而非document

object.getAttribute(属性的名字)

object.setAttribute(属性，值),

childNodes属性可以获取任何一个元素的子节点：element.childNodes

获取节点的nodeType属性：node.nodeType 元素节点的nodeType值是1，属性节点的nodeType值是2 文本节点的nodeType值是2

得到一个节点的值：node.nodevalue

FirstChild第一个子节点 lastChild最后一个子节点

This可以用来代表任何一种当前元素

eg:<a href=” ” onclick=”pop(this.href);return false;” >Example</a>

只有正确地使用标记语言才能对内容做出准确的描述

把文档里的所有连接放入一个数组里

遍历数组

如果某个连接的CLASS属性等于POP，就表示这个链接在被点击时应该调用POP()函数

把这个链接的href属性值传递给POP（）函数

取消这个链接的默认行为，不让这个链接把访问者带离当前窗口

Var links=document.getElenmentByTagName(‘a’)

For (var i=0;i<links.length;i++)

{

If(links[i].getAttribute(“class”)==”pop”)

{ links[i].onclock=function()

{pop(this.getAttribute(“href”));

Return false;

}

}

}

脚本加载时文档可能不完整，所以模型也不完整。没有完整的DOM,getElementsByTagName等方法就不能正常工作，所以window.onload=prepareLinks(当window对象触发onload事件时，document对象已经存在)

几乎所有东西，包括各种方法在内（要去掉后面圆括号），都可以被当做对象来对待

If(!document.getElementById||!document.getElementsByTagName) return false;

性能考虑：

1. 尽量少访问DOM（把一次搜索的结果保存在一个变量里，在多个函数都会取得一组类似元素的情况下，可以考虑把搜索结果保存在一个全局变量里）和尽量少标记（减少文档中的标记数量）
2. 2.减少请求数量（把各个函数放在一个脚本文件里）
3. 把所有<script>标签都放到文档的末尾，</body>标签之前，就可以让页面变得更快
4. 所谓压缩脚本，指的是把脚本文件中不必要的字节，如空格和注释，统统删掉
5. 如果把函数逐一绑到onload事件上，只有最后一个函数才能被执行，可以创建一个匿名函数来容纳这两个函数，然后把那个匿名函数绑定到onload事件上，window.onload=function()

{

firstFunction();

secondFunction()

}

1. 在onclick事件处理函数中，可以用！对showPic的返回值进行取反

VUE.JS

Vue.js 的核心是一个允许你采用简洁的模板语法来声明式的将数据渲染进 DOM 的系统：

**Vue属性：el date computed methods**

**三 ： 具体指令**

**1. v-text**

作用 ： 操作元素中的纯文本

快捷方式 ： {{}}

{{}}里边支持表达式，但不是所有的表达式都能放在里边，只有单个语句的可以，像var和if就不可以，如果想使用if语句的话，用三元运算符代替

<h1 id="app1" v-text="'今天是'+year+'年'+month+'月'"></h1>（这里v-text=""，双引号并不是代表字符串，而是vue自定义的划定界限的符号。如果我们里边输出字符串，就要在里边再添加一对单引号。）

**2. v-html**

作用 ： 操作元素中的HTML标签

v-text会将元素当成纯文本输出（不能解析html元素，只会照样输出），v-html会将元素当成HTML标签解析后输出

**3. v-bind**

参数在指令后以冒号指明。

<a v-bind:href="url"> (缩写<a :href="url"></a>)

<div v-bind:class="{active: isActive}"></div> 这里 isActive 的值为 true 则会被渲染成 class=‘active‘ 如果为 false , 则 不会存在。

v-bind:class="{ active: isActive, 'text-danger': hasError }">

4**. v-on**

按钮的事件我们可以使用 v-on 监听事件，并对用户的输入进行响应。：<a v-on:click="doSomething">缩写为：<a @click="doSomething"></a>

5**.v-model**

用来在 input、select、text、checkbox、radio 等表单控件元素上创建双向数据绑定：

<div id="app">

<p>{{ message }}</p>

<input v-model="message"> </div>

<script> new Vue({ el: '#app', data: { message: 'Runoob!' } })</script>

**6.v-show**

使用 v-show 指令来根据条件展示元素：

<h1 v-show="ok">Hello!</h1>

**7，循环语句**

v-for表示循环：循环data里的内容

8.过滤器

过滤器函数接受表达式的值作为第一个参数。

{{ message | capitalize }}过滤器可以串联：{{ message | filterA | filterB }}

过滤器是 JavaScript 函数，因此可以接受参数：{{ message | filterA('arg1', arg2) }}

9.条件语句

**v-if**

表示是否插入这个元素，true表示是。

**v-else**

可以用 v-else 指令给 v-if 添加一个 "else" 块：

**v-else-if**

用作 v-if 的 else-if 块。可以链式的多次使用：

<div id="app">

<div v-if="Math.random() > 0.5"> Sorry</div>

<div v-else-if> ok </div>

<div v-else-if> ook</div>

<div v-else> Not sorry</div>

</div>

<script>

new Vue({

el: '#app'

})

</script>

</body>

**循环语句**

循环使用 v-for 指令。

v-for 指令需要以 **site in sites** 形式的特殊语法， sites 是源数据数组并且 site 是数组元素迭代的别名。v-for 可以绑定数据到数组来渲染一个列表：

<div id="app">

<ol>

<li v-for="site in sites">

{{ site.name }}

</li>

</ol>

</div>

<script>

new Vue({

el: '#app',

data: { sites: [ { name: 'Runoob' },

{ name: 'Google' },

{ name: 'Taobao' }

]

}

})

</script>

<div id="app">

<ul>

<li v-for="value in object">

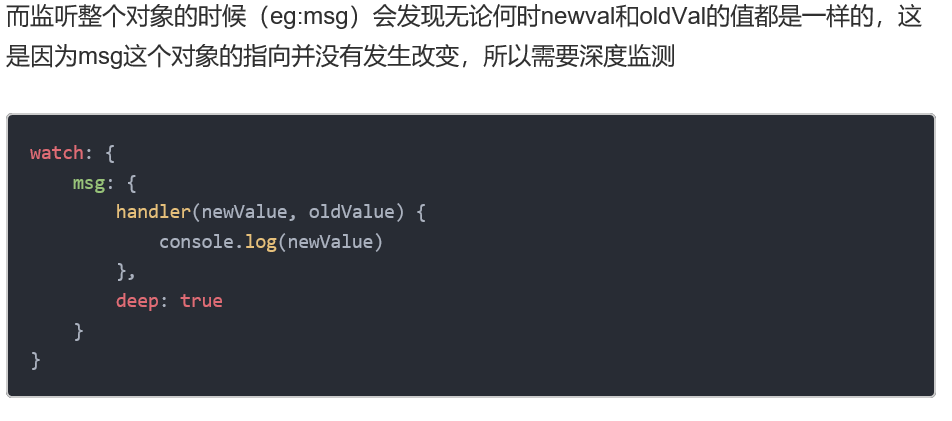
{{ value }}

</li>

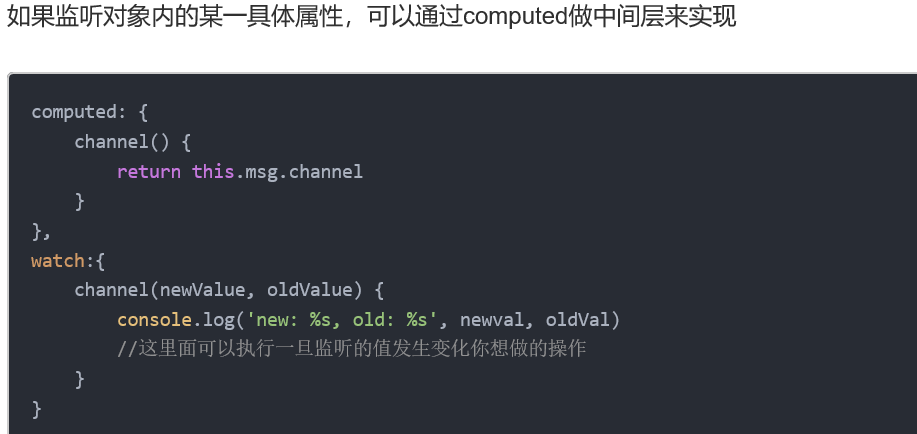
</ul>

</div>

**监测属性watch**



但是如果你要监听的数据是对象内的某一属性，直接watch对象的属性(eg:msg.channel)就会报错



**自定义指令**

使用vue.drective()来全局注册指令

**组件**

组件是可复用的 Vue 实例，且带有一个名字

注册一个全局组件格式如下：Vue.component(tagName, options) options 为配置选项。

全局注册的组件可以用在其被注册之后的任何 (通过 new Vue) 新创建的 Vue 根实例，也包括其组件树中的所有子组件的模板中。

**一个组件的 data 选项必须是一个函数**，因此每个实例可以维护一份被返回对象的独立的拷贝：

data: function () {

return {

count: 0

}

}

Vue.component('blog-post', {

**props: ['title'],**

template: '<h3>{{ title }}</h3>'

})

一个 prop 被注册之后，你就可以像这样把数据作为一个自定义特性传递进来：<blog-post title="My journey with Vue"></blog-post>

