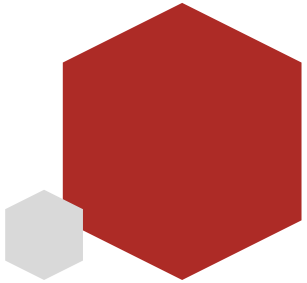
Git&黑马就业数据平台

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌



认识及安装Git

认识及安装Git

版本控制系统: 版本控制是一种记录一个或若干文件内容变化，以便将来查阅特定版本修订情况的系统。

作用:

1. 记录（项目）文件变化
2. 查看记录信息
3. 将文件切换到记录时的状态

安装Git

步骤-windows:

1. 根据操作系统类型下载安装包
2. 双击安装
3. 全部使用默认设置
4. 通过鼠标右键确认安装结果

步骤-mac:

1. 下载安装包
2. 双击安装（调整安全性设置）
3. 终端输入 git --version 确认安装

总结

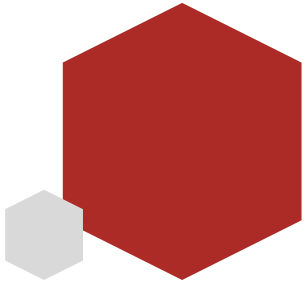
1. Git是一个版本控制系统

2. 版本控制系统：

   记录，查看，切换

3. 如何确认Git安装成功：


```
hulinghao@hulinghaodeMBP2 ~ % git --version
git version 2.15.0
hulinghao@hulinghaodeMBP2 ~ %
```



Git配置用户信息

Git配置用户信息

文档地址: 安装完 Git 之后，要做的第一件事就是设置你的用户名和邮件地址。 因为每一个 Git 提交都会使用这些信息

配置命令:

1. git config --global user.name "用户名"
2. git config --global user.email 邮箱地址
3. **注:** 使用提前注册好的Gitee的用户名和邮箱

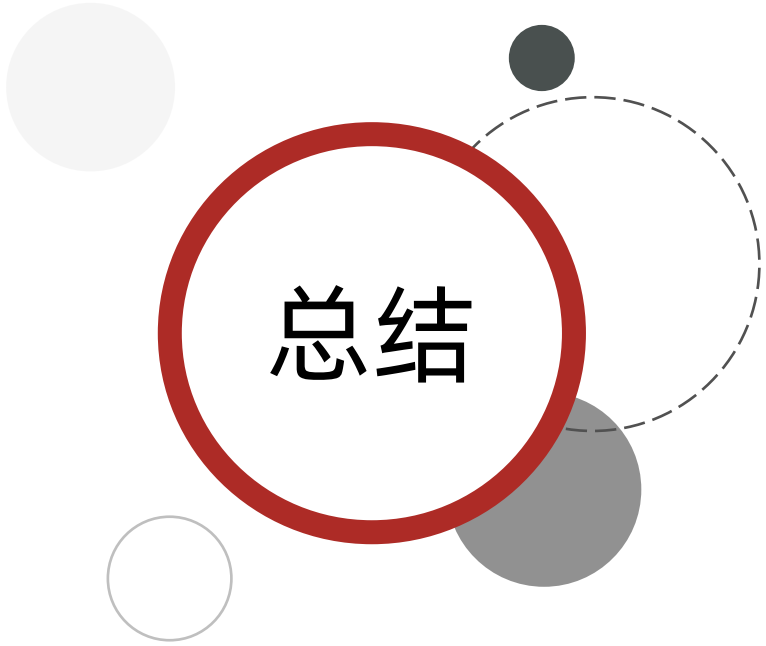
查看配置:

1. git config --list
2. **注:** 信息太多可以输入 q 退出

输入方式:

1. windows: 打开 git bash 输入
2. mac: 打开 终端 输入

```
commit faf8d7574bf9a08e0117188bf12084600b9ea387 (HEAD -> main)
Author: autumnfish <517729329@qq.com>
Date:   Tue Mar 14 10:41:59 2023 +0800

    添加js并整合到首页

commit 5facf79e1581c95f76fa5282825d3c35dc29fe67
Author: autumnfish <517729329@qq.com>
Date:   Tue Mar 14 10:41:03 2023 +0800

    添加样式并整合到首页

commit c9f1acb59afe01a2eedd300cf81ef9218b11cc07
Author: autumnfish <517729329@qq.com>
Date:   Tue Mar 14 10:39:19 2023 +0800

    创建首页并添加默认结构
```

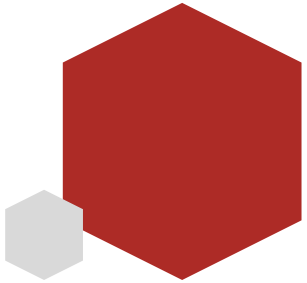


1. Git配置用户信息

   git config --global user.name "用户名"
   git config --global user.email 邮箱地址

2. 查看配置

   git config --list
   信息太多使用 输入 q 退出



本地初始化Git仓库

获取Git仓库

文档地址: 通常有两种获取 Git 项目仓库的方式

1. 将 尚未进行版本控制 的本地目录 转为 Git仓库（初始化仓库）
2. 从其他服务器 克隆 一个已存在的Git仓库

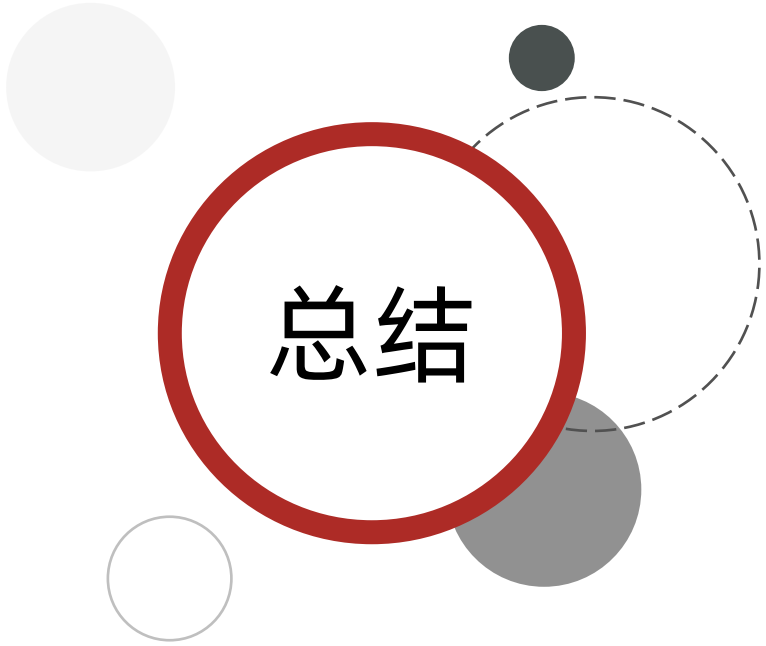
本地初始化Git仓库

步骤：

1. 创建新文件夹并在该文件夹下打开：
    1. windows: git bash
    2. mac: 终端
2. 输入命令 git init

注意：

1. windows: 需要设置显示隐藏文件
2. mac: 快捷键 Command + Shift + . 切换隐藏文件显示



1. 通常有两种获取Git仓库的方式：

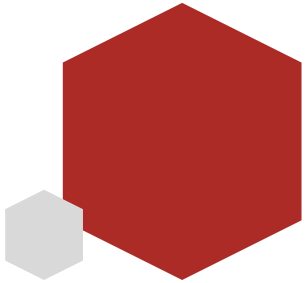
   将 尚未进行版本控制 的本地目录 转为 Git仓库（初始化仓库）
   从其他服务器 克隆 一个已存在的Git仓库

2. 本地初始化Git仓库的步骤：

   在本地目录打开git bash（终端）
   执行命令 git init

3. 如何显示 .git 隐藏文件夹：

   windows: 设置显示隐藏文件
   mac: Command + Shift + .



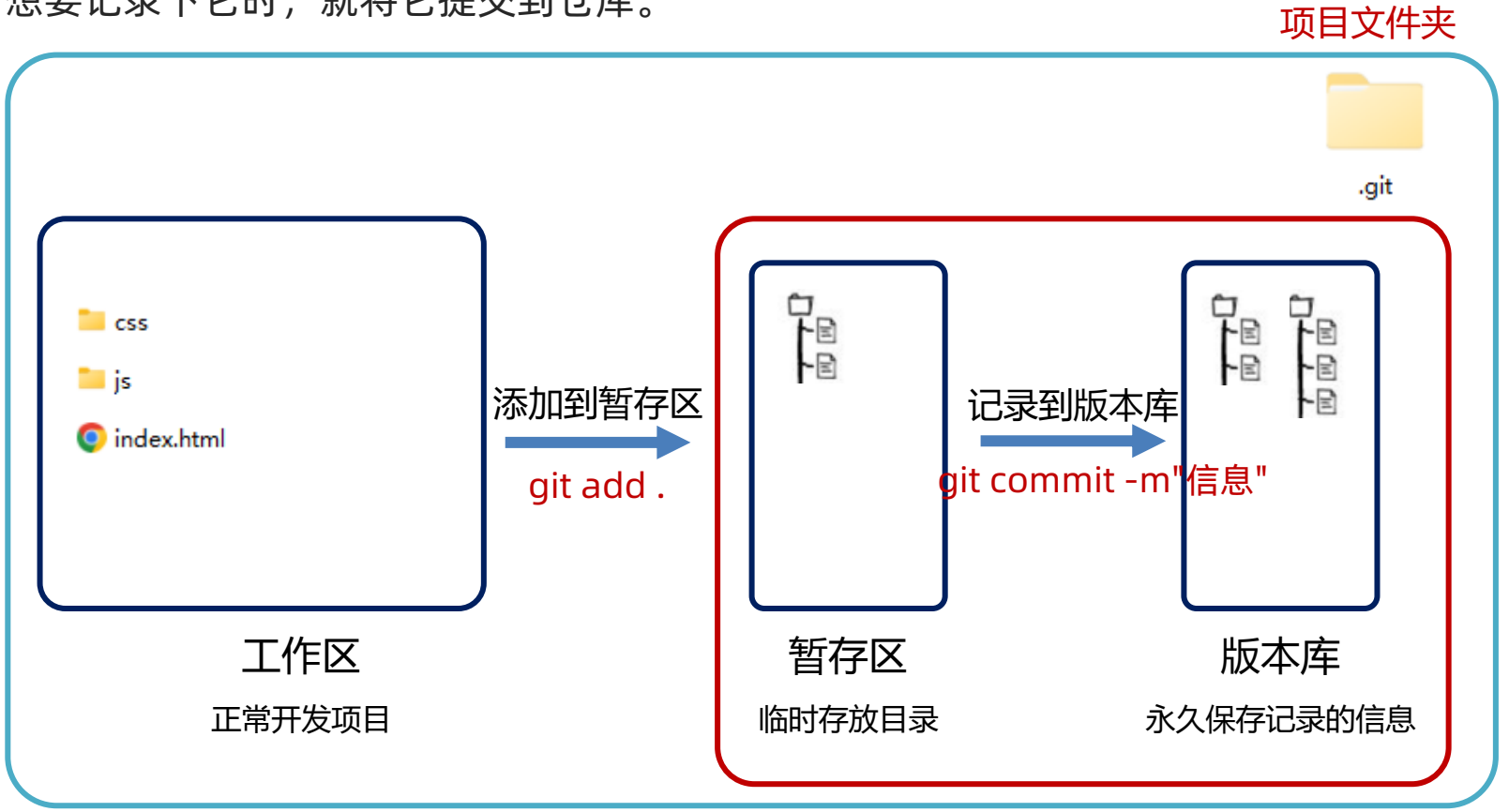
记录每次更新到仓库

记录每次更新到仓库

文档地址:每当完成了一个阶段的目标，想要记录下它时，就将它提交到仓库。

核心操作:

1. 工作区开发
2. 将修改后的文件添加到暂存区
3. 将暂存区的文件记录到版本库



记录每次更新到仓库

核心操作及命令：

1. 工作区开发

   项目文件夹

2. 将修改后的文件添加到暂存区

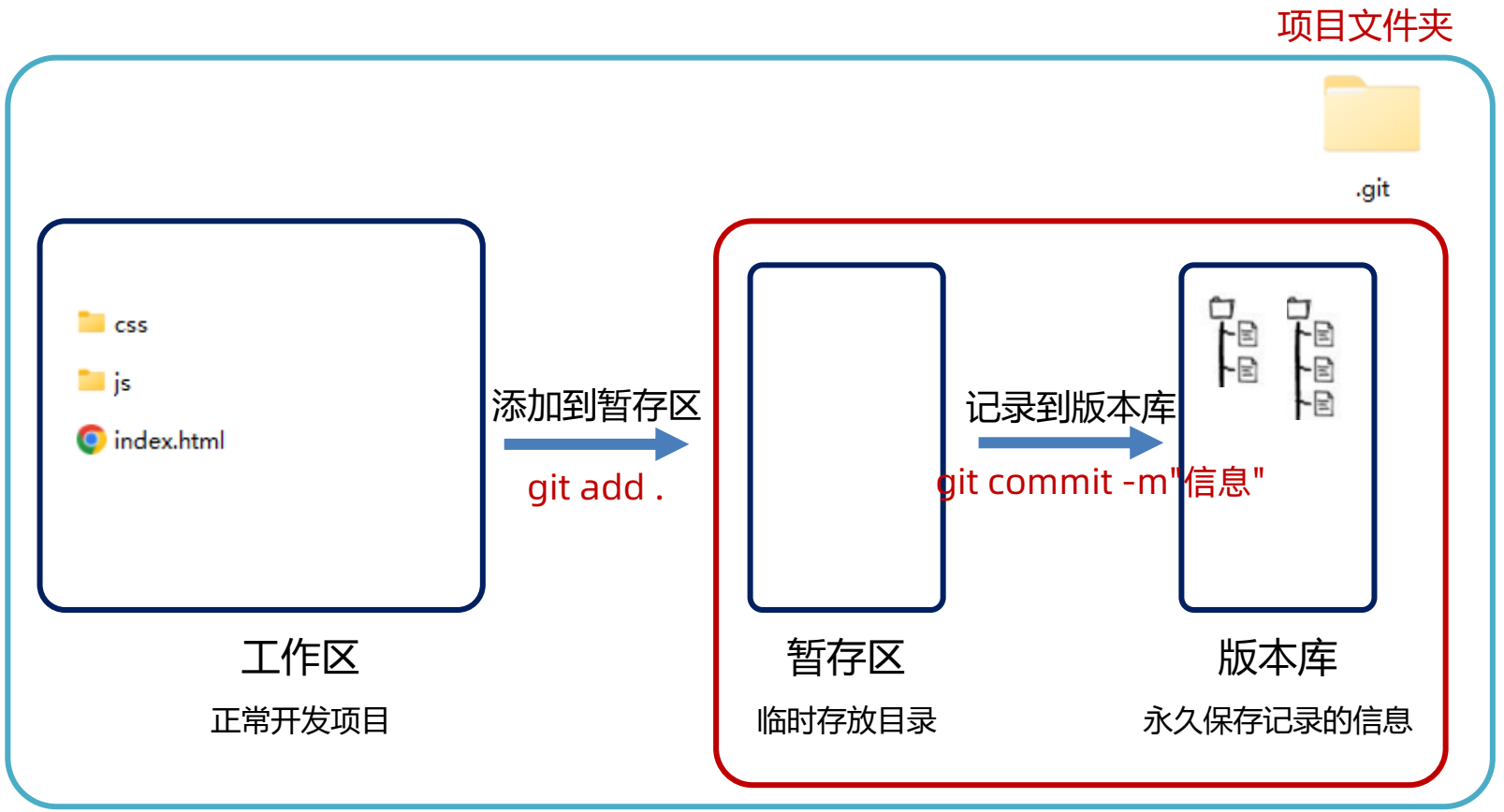
   git add .

3. 将暂存区的文件记录到版本库

   git commit -m"信息"

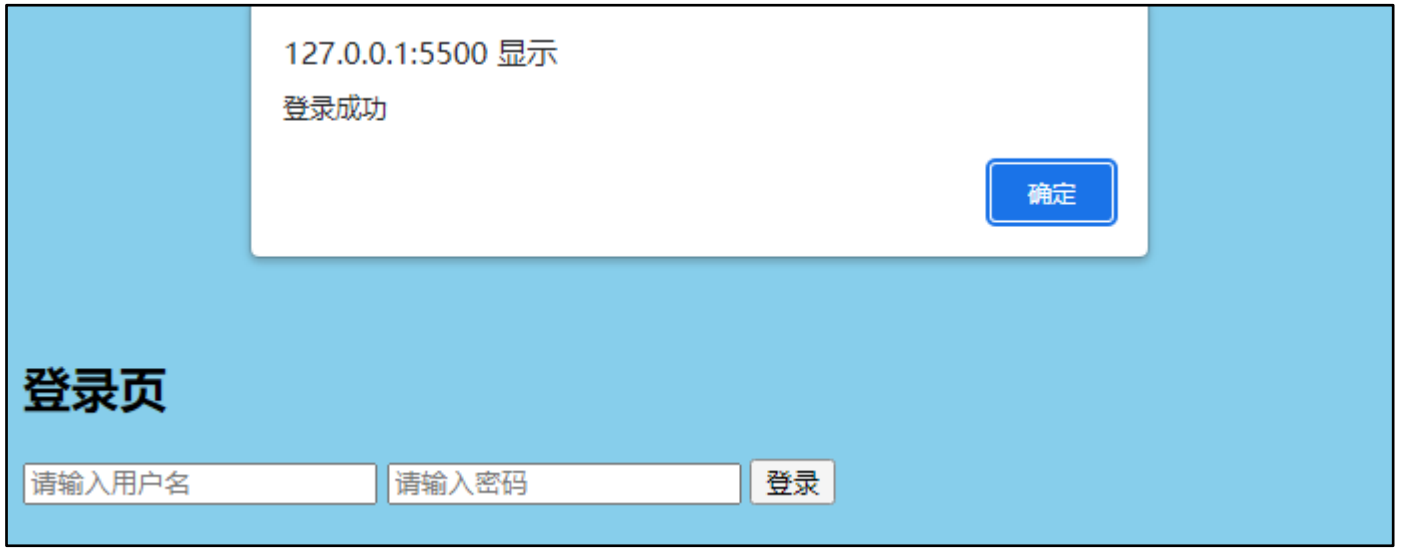
注意：

在项目文件夹下打开 git bash（终端）



需求

实现页面效果，并用Git记录每一次操作

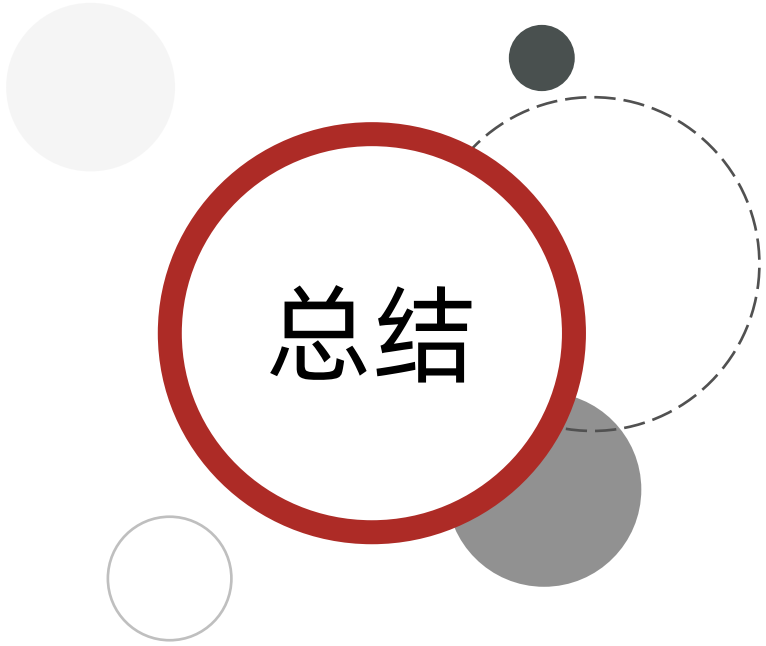


参考步骤：

1. 创建新文件夹并初始化仓库
2. 创建首页并添加结构（Git记录）
3. 创建css并编写样式（Git记录）
4. 创建js并编写逻辑（Git记录）

核心操作：

工作区开发 → 添加到暂存区 git add .→ 记录到版本库 git commit -m"信息"



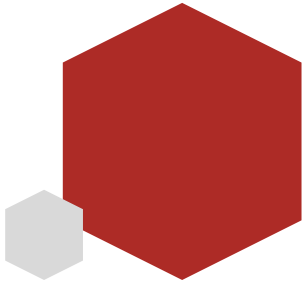
1. 使用Git进行版本控制的项目中的3个区：

   工作区、暂存区、版本库

2. 记录每次更新到仓库的核心操作：

   工作区开发→添加到暂存区 →记录到版本库

   工作区开发→ git add . → git commit -m"信息"



查看及切换历史版本

查看及切换历史版本

核心操作及命令：

1. 查看历史版本

   git log --oneline

   git log

2. 切换历史版本

   git reset --hard 版本号

拓展命令：

1. git bash（终端）清屏:

   clear

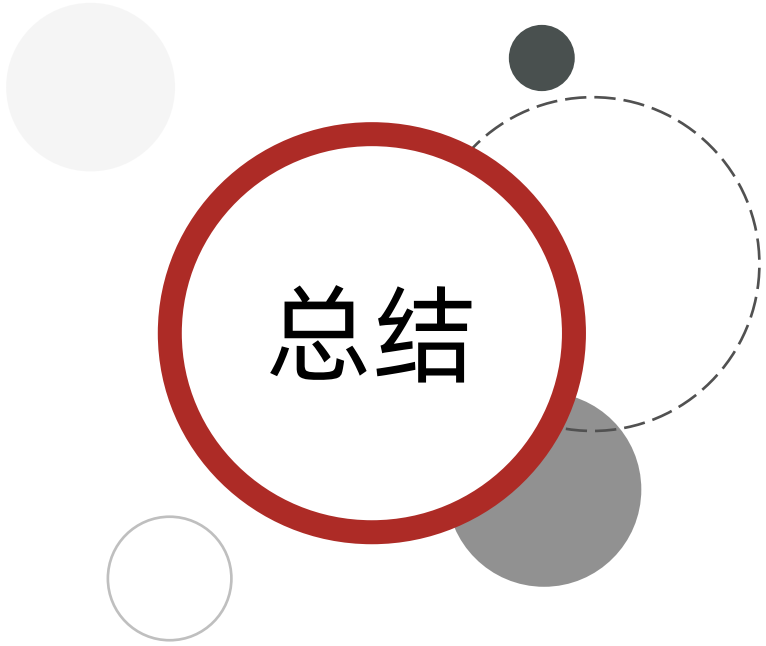
2. 查看完整历史

   git reflog

```
1315c35 (HEAD -> main) 创建js并编写逻辑
79d2ce5 创建css并编写样式
be3769b 创建首页并添加结构
```

测试：

1. 查看上一节项目记录的历史版本
2. 切换历史版本



1. 查看历史版本的命令：

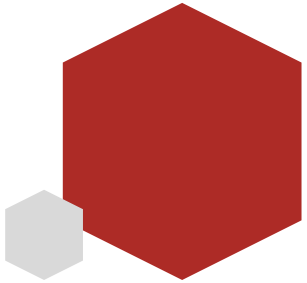
   git log --oneline 、 git log

2. 切换历史版本的命令：

   git reset --hard 版本号

3. 拓展命令：

   clear（清屏）、git reflog（查看完整历史）



Git忽略文件和查看文件状态

Git忽略文件

文档地址: 我们总会有些文件无需纳入 Git 的管理，也不希望它们总出现在未跟踪文件列表。 通常都是些自动生成的文件，比如日志文件，或者编译过程中创建的临时文件等。 在这种情况下，我们可以创建一个名为 .gitignore 的文件，列出要忽略的文件

```
# 这里演示的部分语法
# #之后的内容是注释 会被Git忽略
# 忽略 info.txt 文件
info.txt

# 忽略 .vscode/ 目录下所有的文件
.vscode

# 忽略目录下所有.md结尾的文件
*.md

# 会忽略 doc/目录下扩展名为txt的文件
doc/*.txt
```

检查文件状态

文档地址:可以用 git status 命令查看哪些文件处于什么状态。

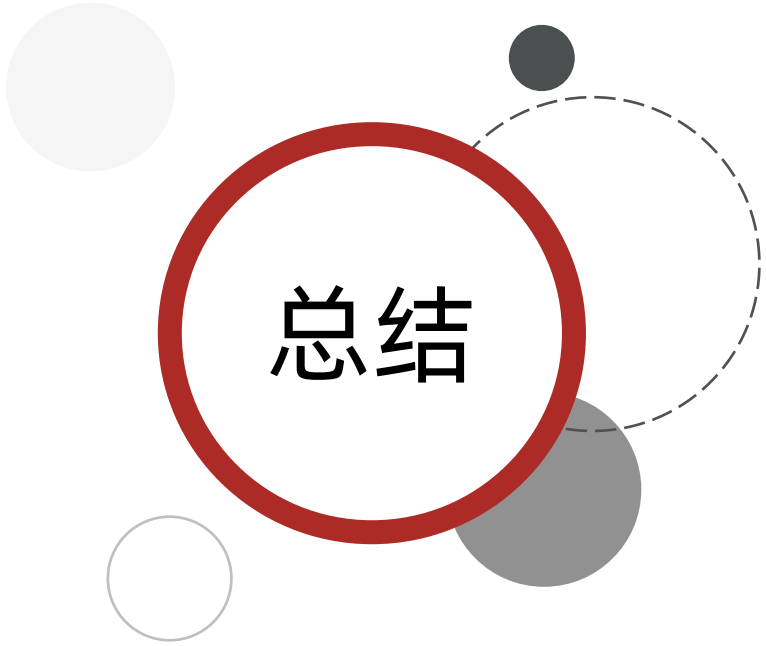
1. 红色：工作区有文件更改
2. 绿色：暂存区有文件更改
3. nothing to commit: 没有任何文件更改

需求

修改VSCode工作区设置，设置失去焦点时自动保存文件，并通过 .gitignore 让Git忽略工作区设置

参考步骤：

1. 创建新文件夹并初始化仓库，VSCode打开文件夹（git status）
2. 调整VSCode工作区设置，开启失去焦点自动保存文件并测试（ git status ）
3. 创建 .gitignore 并配置忽略生成的工作区设置（ git status ）
4. 使用Git记录，记录的过程中查看文件状态



1. Git设置忽略文件需要创建的文件是:

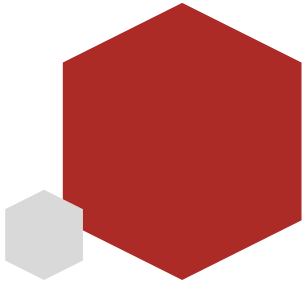
   .gitignore

2. .gitignore文件的常用语法:

```
# 这里演示的部分语法
# #之后的内容是注释 会被Git忽略
# 忽略 info.txt 文件
info.txt

# 忽略 .vscode/ 目录下所有的文件
.vscode

# 忽略目录下所有.md结尾的文件
*.md

# 会忽略 doc/目录下扩展名为txt的文件
doc/*.txt
```

3. 查看文件状态的命令是: git status

   红色、绿色、nothing to commit

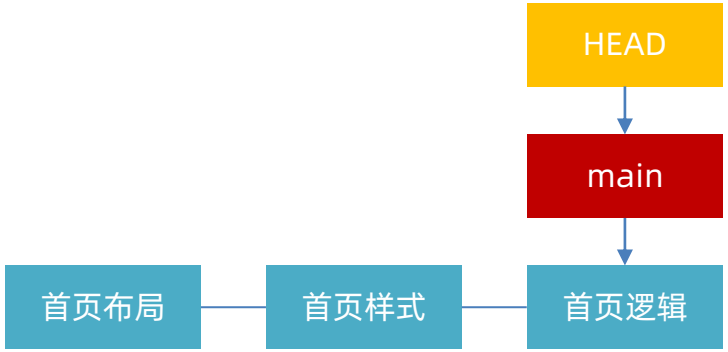


Git分支-查看及切换

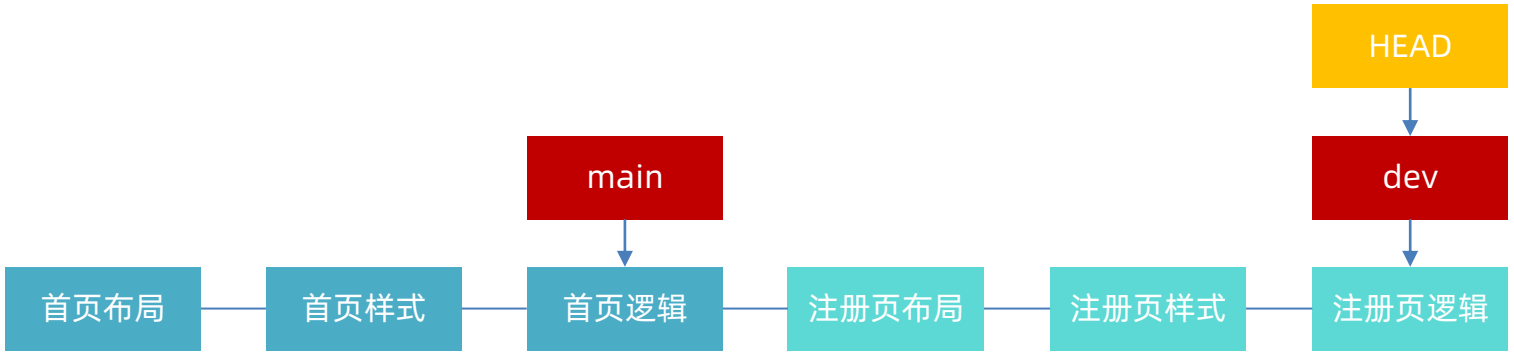
Git分支

文档地址: 使用分支意味着你可以把你的工作从开发主线上分离开来，以免影响开发主线。

```
d6e18e4 (HEAD -> main) 首页逻辑
b38ecbf 首页样式
c77ff64 首页布局
```




```
f488d4b (HEAD -> dev) 注册页逻辑
c6287d0 注册页样式
b701a92 注册页结构
d6e18e4 (main) 首页逻辑
b38ecbf 首页样式
c77ff64 首页布局
```

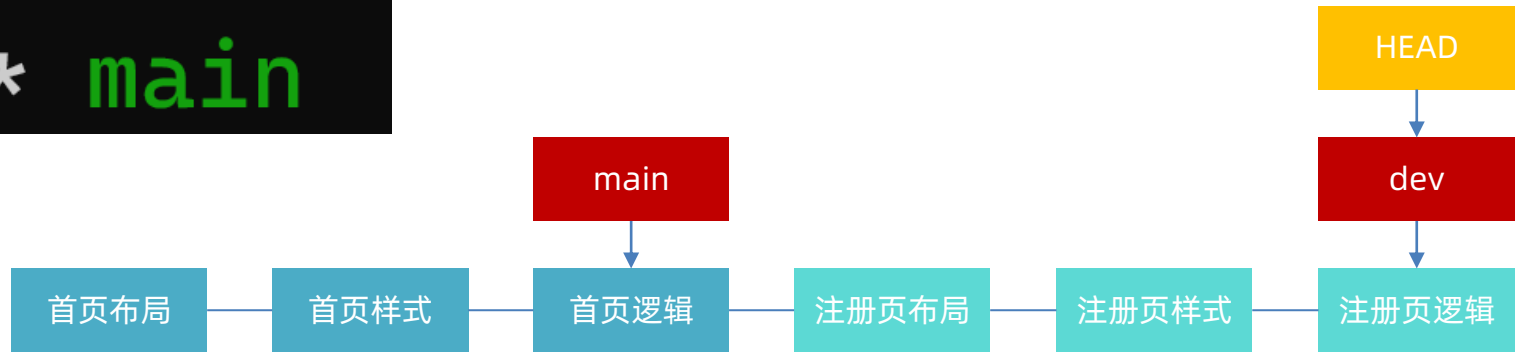


注意: Git初始化仓库之后默认使用的分支名是 main（早期是 master），默认分支名不相同，不影响后续操作

Git分支-常用操作

操作	命令
查看分支	git branch
创建分支	
切换分支	git checkout 分支名
合并分支	
删除分支	
重命名分支	git branch –m 老分支名 新分支名

```
  dev
* main
```

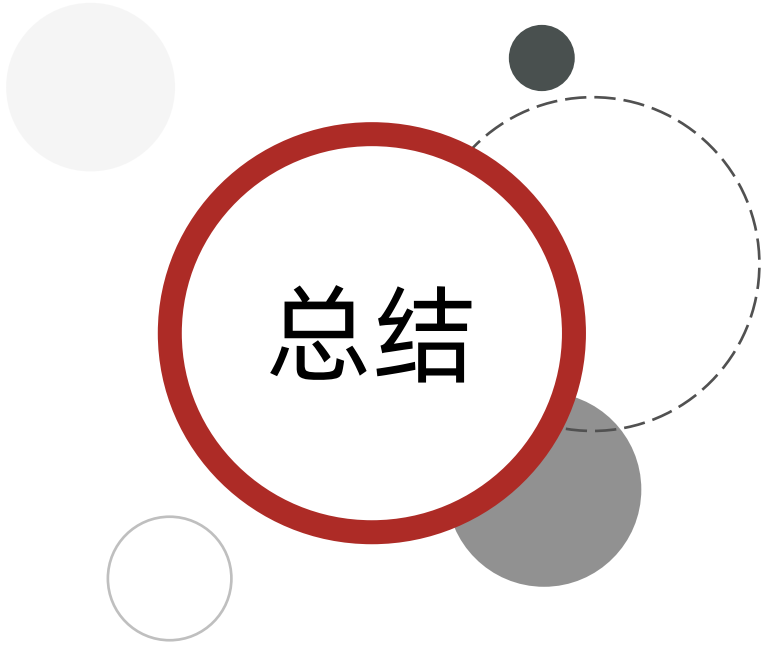


注意: 命令执行的位置，如非特殊说明，均为项目根目录

需求

基于提供的Git仓库，测试 查看分支 和 切换分支 的命令

操作	命令
查看分支	git branch
创建分支	
切换分支	git checkout 分支名
合并分支	
删除分支	
重命名分支	git branch –m 老分支名 新分支名

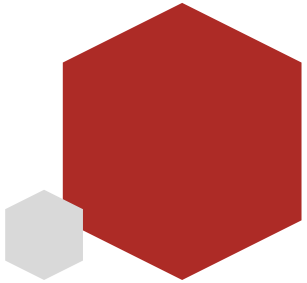


1. Git分支：

   把工作从开发主线分离，以免影响开发主线

2. 操作命令：

操作	命令
查看分支	git branch
创建分支	
切换分支	git checkout 分支名
合并分支	
删除分支	
重命名分支	git branch –m 老分支名 新分支名

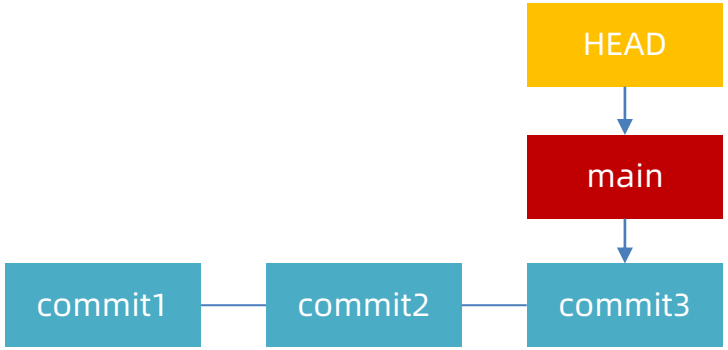


Git分支-创建分支

创建分支

创建分支就是创建了一个新的可以移动的指针，默认的指向和原分支一样

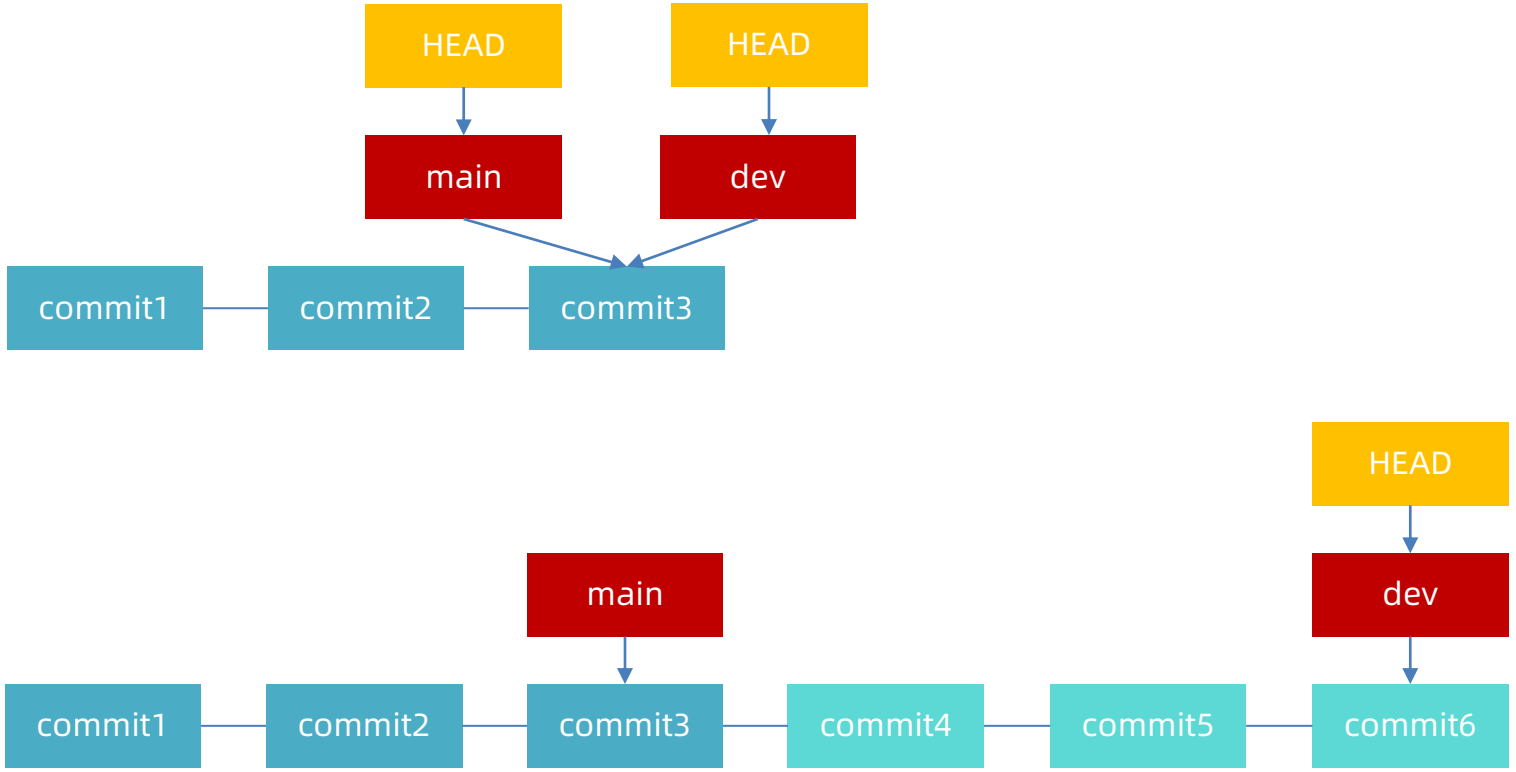
操作	命令
查看分支	git branch
创建分支	git branch 新分支名
切换分支	git checkout 分支名
合并分支	
删除分支	
重命名分支	git branch –m 老分支名 新分支名



创建分支

创建分支就是创建了一个新的可以移动的指针，默认的指向和原分支一样

操作	命令
查看分支	git branch
创建分支	git branch 新分支名
切换分支	git checkout 分支名
合并分支	
删除分支	
重命名分支	git branch –m 老分支名 新分支名



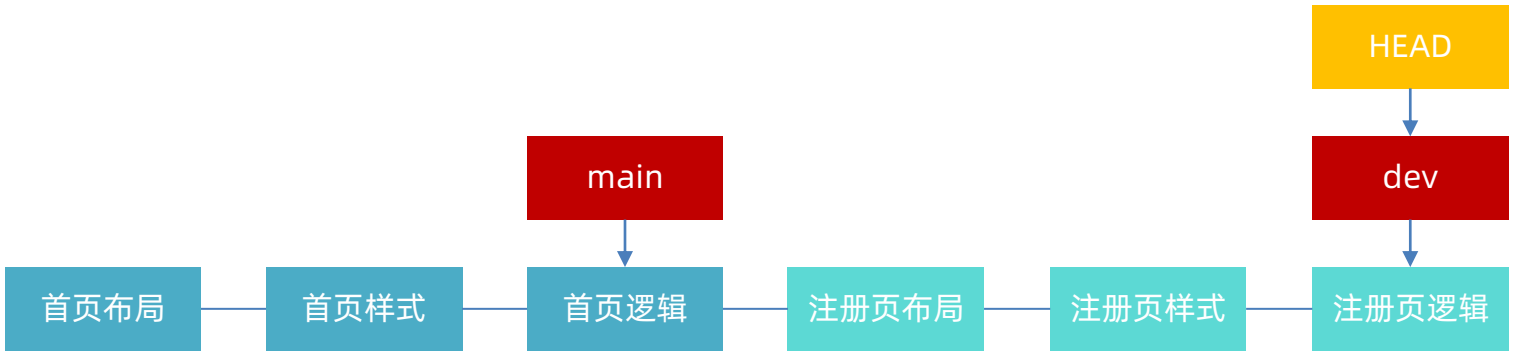
需求

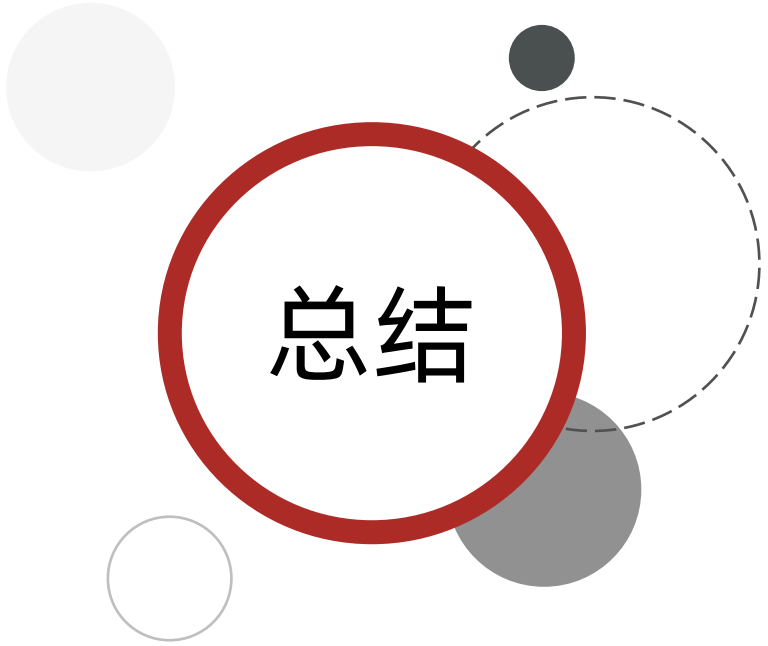
基于提供的Git仓库，创建并切换到新的分支（dev），并实现功能

参考步骤：

1. 创建并切换分支（dev）
2. 实现注册页布局、样式、逻辑，并记录

操作	命令
查看分支	git branch
创建分支	git branch 新分支
切换分支	git checkout 分支名
合并分支	
删除分支	
重命名分支	git branch –m 老分支名 新分支名



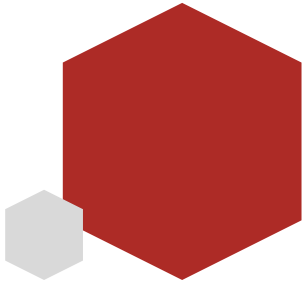


1. 创建分支：

   创建一个新可以移动的指针，默认指向和原分支相同

2. 操作命令：

操作	命令
查看分支	git branch
创建分支	git branch 新分支名
切换分支	git checkout 分支名
合并分支	
删除分支	
重命名分支	git branch –m 老分支名 新分支名



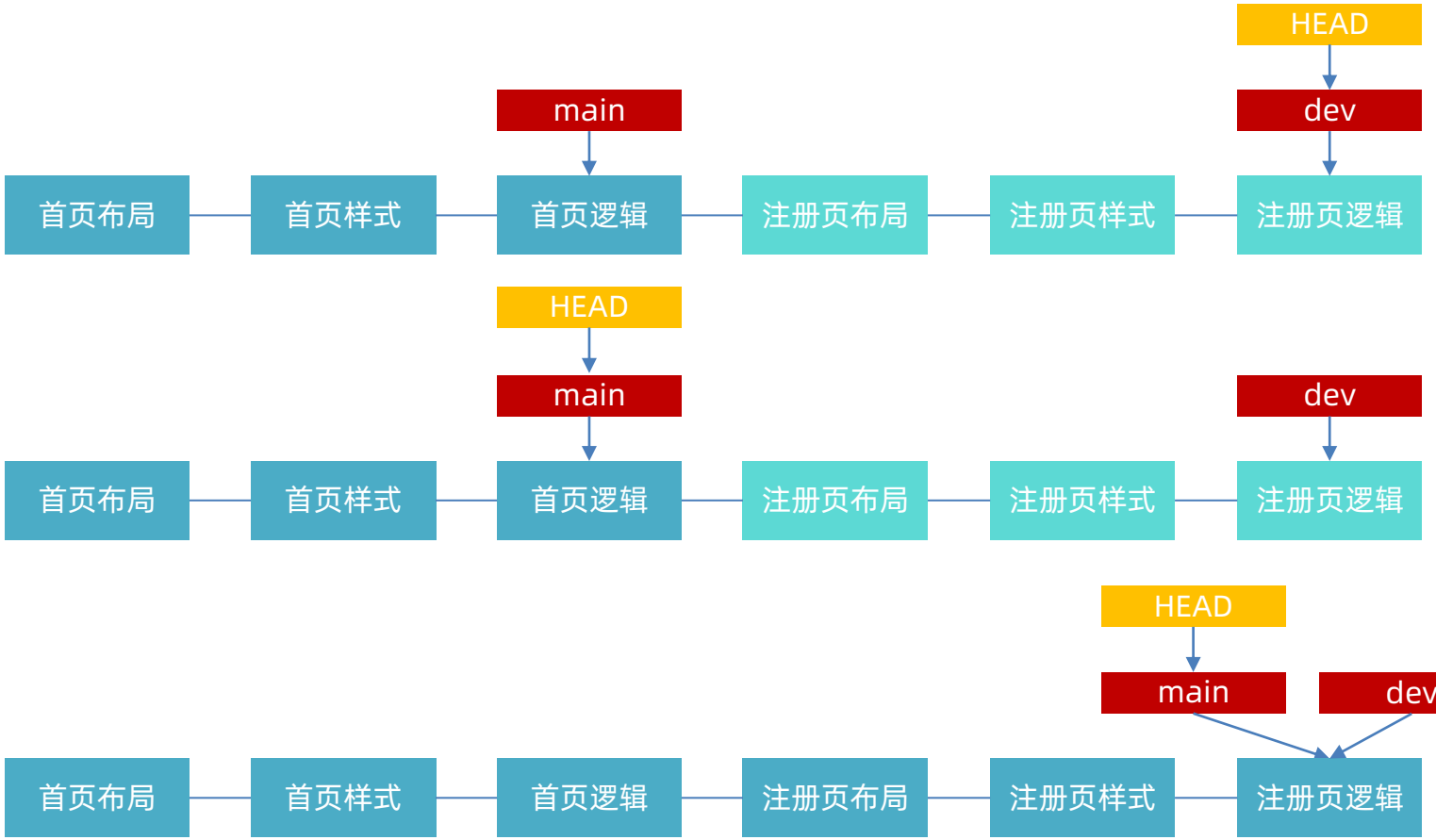
Git分支-合并及删除分支

合并及删除分支

合并分支可以将某个分支上的所有commit，并到当前分支的commit

合并完毕之后，可以删除多余分支

操作	命令
查看分支	git branch
创建分支	git branch 新分支名
切换分支	git checkout 分支名
合并分支	git merge 分支名
删除分支	git branch –d 分支名
重命名分支	git branch –m 老分支名 新分支名



需求

将上一节Git仓库中的 dev 分支 合并到 main 分支，并删除 dev 分支

参考步骤:

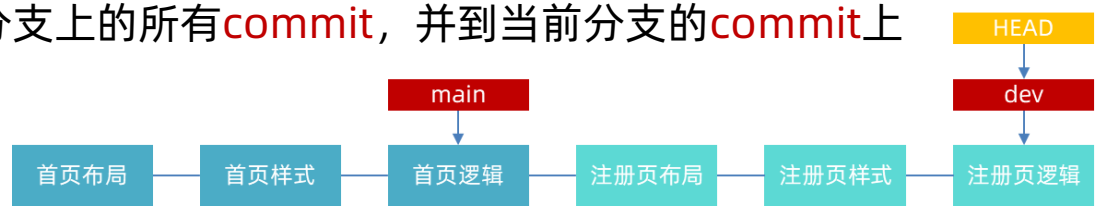
1. 切换到main分支
2. 合并dev分支
3. 删除dev分支

操作	命令
查看分支	git branch
创建分支	git branch 新分支名
切换分支	git checkout 分支名
合并分支	git merge 分支名
删除分支	git branch –d 分支名
重命名分支	git branch –m 老分支名 新分支名

总结

1. Git合并分支：

   某个分支上的所有commit，并到当前分支的commit上

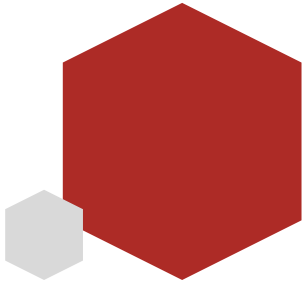


2. 删除分支：

   合并之后，可以删除多余分支

3. 操作命令：

操作	命令
查看分支	git branch
创建分支	git branch 新分支名
切换分支	git checkout 分支名
合并分支	git merge 分支名
删除分支	git branch –d 分支名
重命名分支	git branch –m 老分支名 新分支名



Git分支-命令补充

Git分支-命令补充

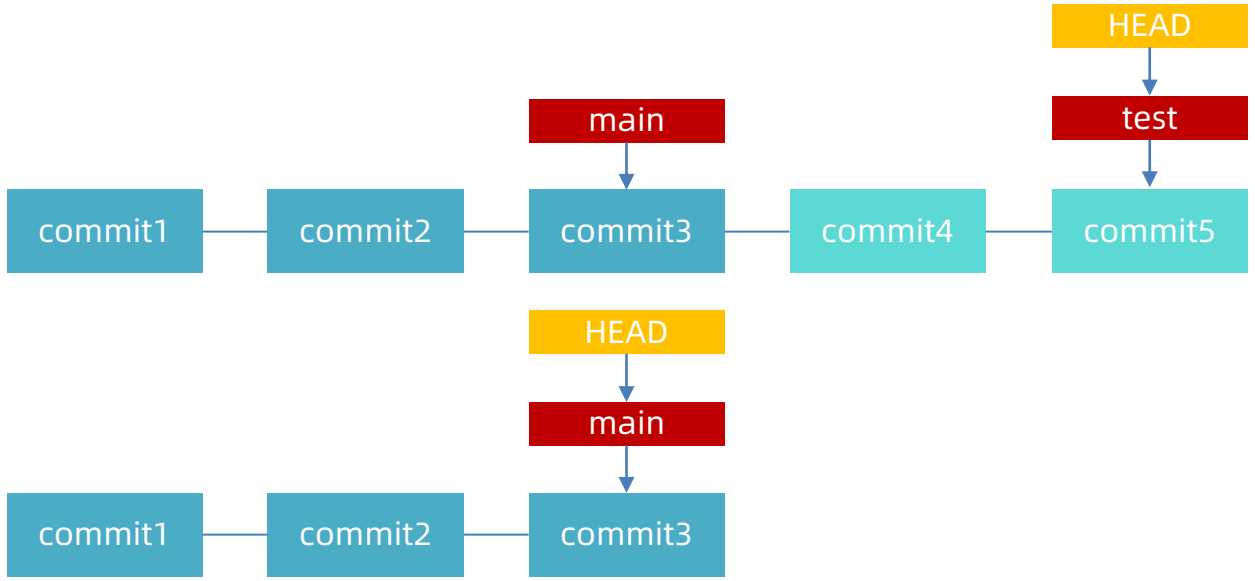
操作	命令
查看分支	git branch
创建分支	git branch 新分支名
切换分支	git checkout 分支名
创建+切换分支	git checkout –b 新分支名
合并分支	git merge 分支名
删除分支	git branch –d 分支名
强制删除分支	git branch –D 分支名
重命名分支	git branch –m 老分支名 新分支名

需求

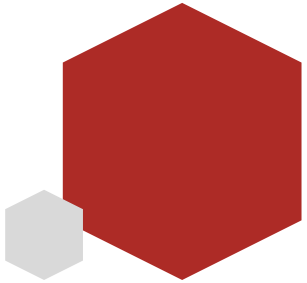
基于提供的Git仓库（main分支+3次commit），测试 补充的命令

参考步骤：

1. 创建并切换到test分支
2. 写2个功能，比如注册页布局、样式并记录
3. 切换到main分支，并强制删除test分支



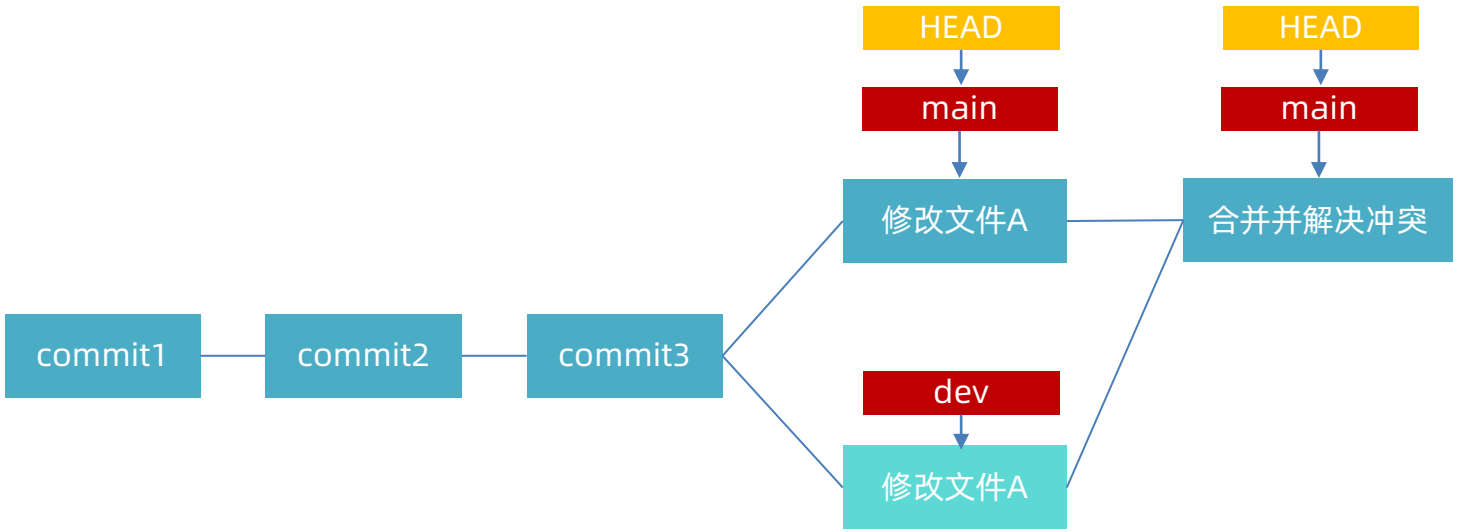
操作	命令
查看分支	git branch
创建分支	git branch 新分支名
切换分支	git checkout 分支名
创建+切换分支	git checkout –b 新分支名
合并分支	git merge 分支名
删除分支	git branch –d 分支名
强制删除分支	git branch –D 分支名
重命名分支	git branch –m 老分支名 新分支名



Git分支-冲突

Git分支-冲突

文档地址: 如果你在两个不同的分支中，对同一个文件的同一个部分进行了不同的修改，Git 就没法干净的合并它们。



需求

基于提供的Git仓库和参考步骤测试合并时出现的冲突，解决冲突并记录

参考步骤:

1. 分别在在dev和main分支修改index.js文件（不同分支，相同文件，相同位置，不同修改）
2. 将dev分支合并到main分支
3. 根据VSCode的提示解决冲突并记录

总结

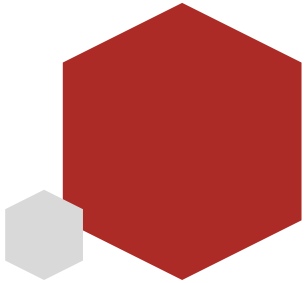
1. 合并分支时为何会出现冲突：

   不同分支，相同文件，相同位置，不同修改

2. 如何根据提示解决冲突：

```
采用当前更改 | 采用传入的更改 | 保留双方更改 | 比较变更
<<<<<<< HEAD （当前更改）
  for (let j = 0; j < 20; j++) {
    console.log('main分支的输出')
=======
  for (let i = 0; i < 10; i++) {
    console.log('dev分支的输出')
>>>>>>> dev （传入的更改）
  }
})
```

3. 解决冲突之后使用Git记录



黑马就业数据平台-项目演示

项目模块

1. 注册业务
2. axios基地址、简化方法
3. toast轻提示抽取

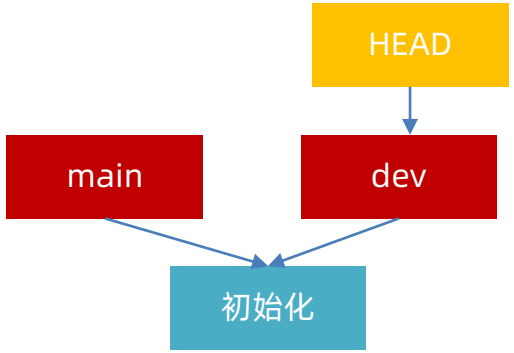
1. 登录业务
2. 登录信息缓存

1. Git远程仓库
2. 页面鉴权
3. axios拦截器
4. echarts图表

1. 数据CRUD

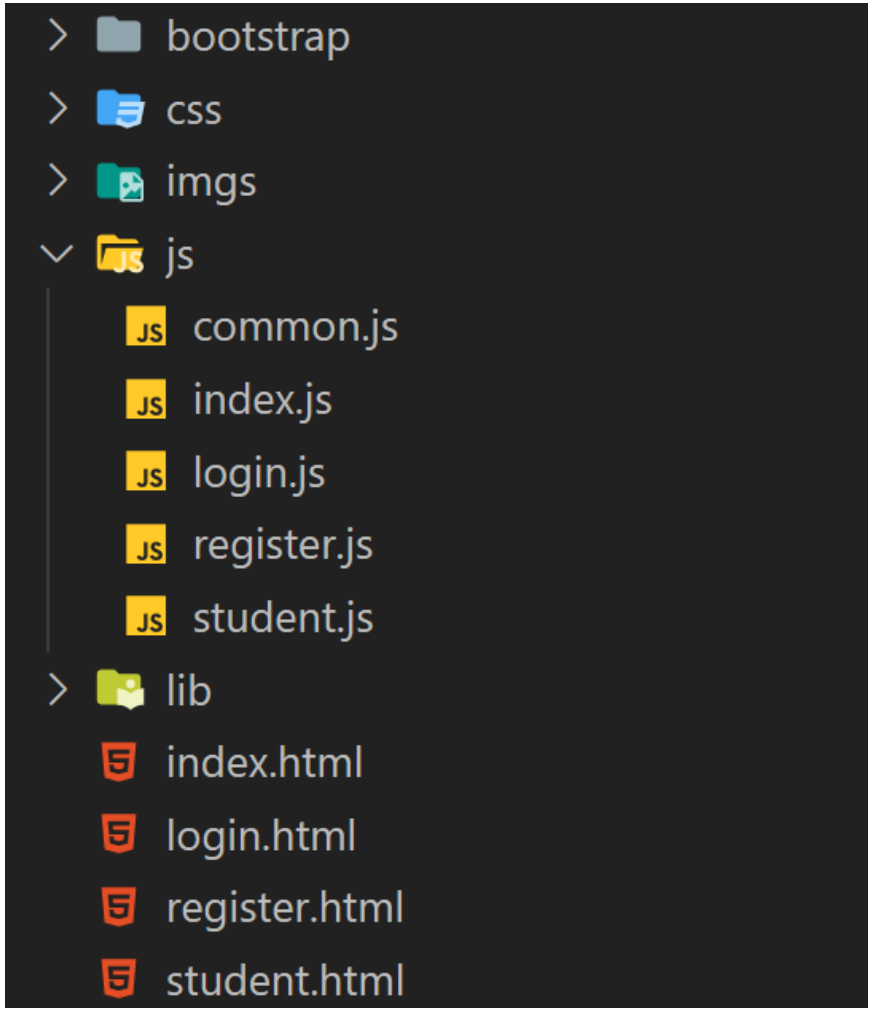
项目初始化

1. 初始化Git仓库，整合提供的模板代码并记录
2. 创建并切换到 dev 分支



项目目录

1. 公共逻辑写在 common.js
2. 页面逻辑写在 js目录下的同名js文件



axios基地址

文档地址: baseURL 将自动加在 url 前面，除非 url 是一个绝对 URL。它可以通过设置一个 baseURL 便于为 axios 实例的方法传递相对 URL

https://hmajax.itheima.net/login
https://hmajax.itheima.net/register
https://hmajax.itheima.net/api/province

```
// 设置基地址
axios.defaults.baseURL = 'https://hmajax.itheima.net'
```

```
// 设置相对URL即可
axios({
  url: '/login'
})
```

```
// 设置相对URL即可
axios({
  url: '/register'
})
```

```
// 设置相对URL即可
axios({
  url: '/api/province'
})
```

配置axios基地址

项目中配置 axios基地址，简化后续URL设置（Git记录）

接口文档地址

配置axios基地址

```
// 设置基地址
axios.defaults.baseURL = 'https://hmajax.itheima.net'
```

测试配置结果

```
axios({
  url: '/register',
  method: 'post',
  data: {
    username: 'itheima666',
    password: '123456'
  }
})
```

抽取轻提示函数

抽取轻提示函数，方便后续调用（Git记录）

抽取轻提示函数

实例化toast

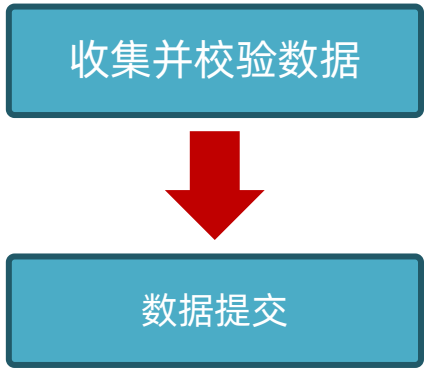
修改内容并显示

```
showToast('注册成功')
```

```
<div class=" toast bg-rgba my-toast" data-bs-delay="1500">
  <div class="toast-body">提示消息</div>
</div>
```

用户注册

完成用户注册功能，提示用户操作结果（Git记录）




```
// axios函数+对象写法
axios({
  url: 'url地址',
  method: 'post',
  data:{key:'value'}
})

// axios别名方法（简化方法）
axios.post('url地址', { key: 'value'})
```

用户登录

完成用户登录功能（Git记录）

收集并校验数据 → 提交数据 → 缓存响应数据 → 跳转首页

传智教育旗下高端IT教育品牌