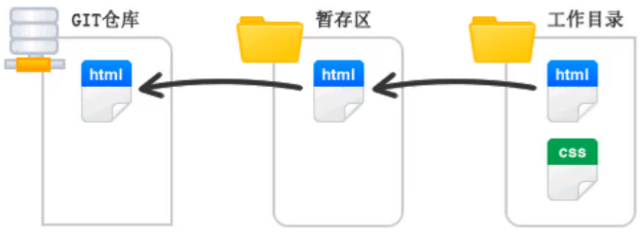
git

什么是git

git就是一个管理版本的工具

1.4 Git 基本工作流程

git仓库	暂存区	工作目录
用于存放提交记录	临时存放被修改文件	被Git管理的项目目录



git的使用

1.5 Git 的使用

1.5.1 Git 使用前配置

在使用 git 前，需要告诉 git 你是谁，在向 git 仓库中提交时需要用到。

1. 配置提交人姓名：`git config --global user.name 提交人姓名`
2. 配置提交人姓名：`git config --global user.email 提交人邮箱`
3. 查看git配置信息：`git config --list`

注意

1. 如果要对配置信息进行修改，重复上述命令即可。
2. 配置只需要执行一次。

1.5.2 提交步骤

1. `git init` 初始化git仓库
2. `git status` 查看文件状态
3. `git add 文件列表` 追踪文件
4. `git commit -m 提交信息` 向仓库中提交代码
5. `git log` 查看提交记录

1.5.3 恢复记录

`git reset --hard commitID` 将 git 仓库中指定的更新记录恢复出来，并且覆盖暂存区和工作目录

设置用户名

在name后面写

`$ git config --global user.name`

设置email

`git config --global user.email`s

查看git配置信息

git config --list

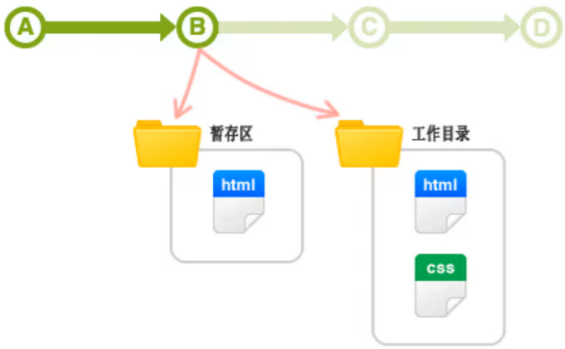
提交步骤

1.5.2 提交步骤

1. `git init` 初始化git仓库
2. `git status` 查看文件状态
3. `git add 文件列表` 追踪文件
4. `git commit -m 提交信息` 向仓库中提交代码
5. `git log` 查看提交记录

1.5.3 恢复记录

`git rest --hard commitID` 将 git 仓库中指定的更新记录恢复出来，并且覆盖暂存区和工作目录。



初始化仓库

`git init`

添加git文件

`git add .`

`git add index.html`

查看文件状态

`git status`

向git仓库提交文件

-m后面务必要写每一次提交的信息以区分

`git commit -m`

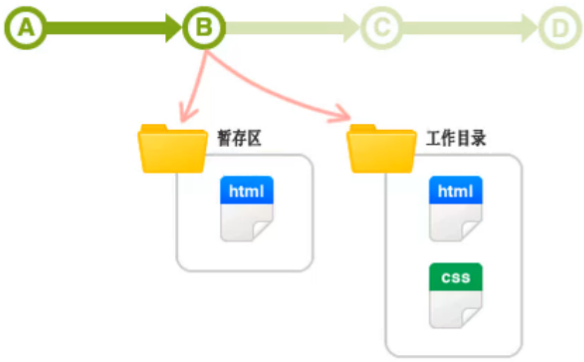
查看提交的历史纪录

`git log`

撤销

1.5.3 撤销

- 用暂存区中的文件覆盖工作目录中的文件：`git checkout 文件`
- 将文件从暂存区中删除：`git rm --cached 文件`
- 将 git 仓库中指定的更新记录恢复出来，并且覆盖暂存区和工作目录：`git rest --hard commitID`



用暂存区域文件覆盖工作目录文件

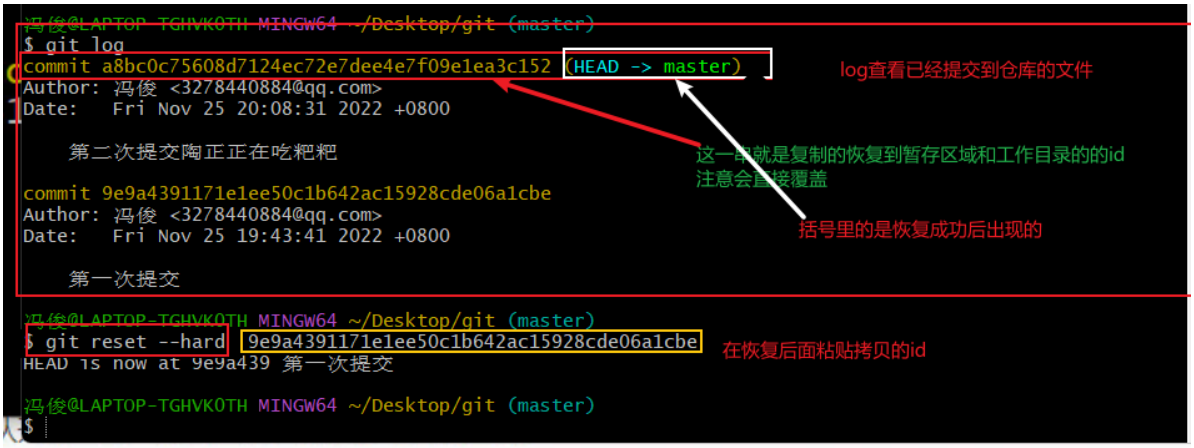
`git checkout list.html`

从暂存区中删除文件

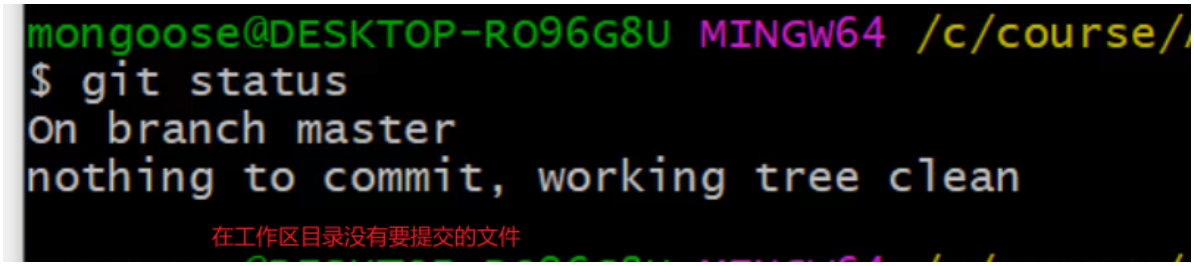
`git rm --cached list.html`

将git仓库中指定的更新记录恢复并且覆盖暂存区域和工作目录

`git reset --hard`



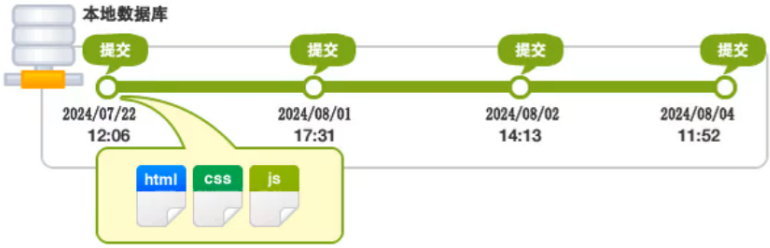
暂存区目录全部提交后



分支

2.1.1 分支细分

1. 主分支（master）：第一次向 git 仓库中提交更新记录时自动产生的一个分支。



2. 、开发分支（develop）：作为开发的分支，基于 master 分支创建。

主分支

主分支：第一次向git仓库中提交更新记录自动产生的一个分支

开发分支

作为开发的分支，基于master

功能分支

作为开发具体功能的分支，基于开发创建分支

> **注：当功能分支累计到一定程度之后就累计到开发分支当开发分支累计到一定程度之后就累计到主分支**

查看分支

2.1.2 分支命令

- `git branch` 查看分支
- `git branch 分支名称` 创建分支
- `git checkout 分支名称` 切换分支
- `git merge 来源分支` 合并分支
- `git branch -d 分支名称` 删除分支（分支被合并后才允许删除） （-D 强制删除）

2.2 暂时保存更改

查看分支

```
git branch
```

返回的master就是主分支

> **如果分支是绿色的且分支前面有*号则表示当前处于此分支，其他分支则是白色表示**

创建分支

基于主分支创建分支在 git branch 后面输入分支名称推荐（develop：开发分支）

```
git branch develop
```

切换分支

```
git checkout develop
```

> 注意：切换分支时当前分支上的分支上的工作一定要提交到git仓库中要保持当前工作区（暂存区）是完全干净的状态

创建并切换分支

```
$ git checkout -b 分支名
```

合并分支

```
git merge develop
```

```
mongoose@DESKTOP-RO96G8U MINGW64 /c/course/AliBaiXiu/day01/code/git-de
$ git merge develop
Updating ecc9675..fa27ec8
Fast-forward
 develop.html | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 develop.html
```

译：1个问题发生变化，0个插入0个删除

删除分支

`git branch -d （已经被合并要删除的分支）`

> - 如果这个分支没有被合并，那么git是不允许删除的
> - 但是可以强制删除

强制删除

`git braner -D(要删除的分支)`

缓存暂保存和更改

储存临时改动：

`git stash`

恢复改动：

`git stash pop`

Github

`$ git push https://github.com/junfengchinese/git-demo.git master`

向github提交，语法push，网址是github上的仓库，后面跟分支

推送命令简单化

git remote add origin https://github.com/junfengchinese/git-demo.git

`git push origin master`

超级简单化

-u记住地址

`git push -u origin master`

下次直接

`git push`

克隆远程仓库

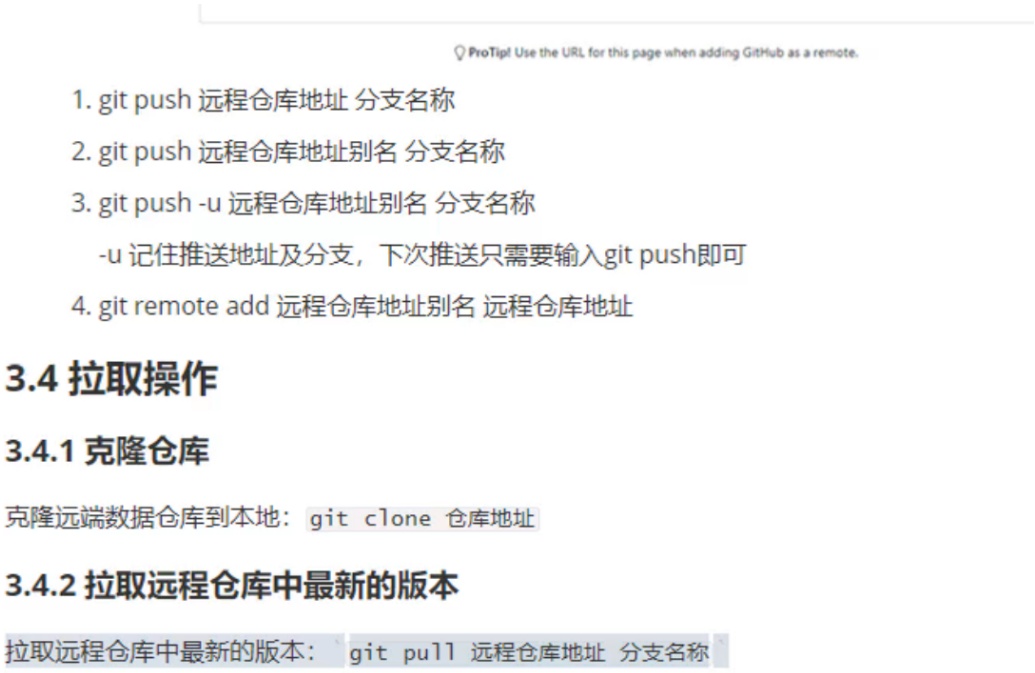
`git clone`

git clone https://github.com/junfengchinese/git-demo.git

其他程序员获取权限后就可以像上面一样上传

拉取远程仓库最新的命令

`get pull` **远程仓库的地址 分支名称**



`git pull origin master --allow-unrelated-histories`

如果合并了两个不同的开始提交的仓库，在新的 git 会发现这两个仓库可能不是同一个，为了防止开发者上传错误，于是就给下面的提示

fatal: refusing to merge unrelated histories

如我在Github新建一个仓库，写了License，然后把本地一个写了很久仓库上传。这时会发现 github 的仓库和本地的没有一个共同的 commit 所以 git 不让提交，认为是写错了 origin ，如果开发者确定是这个 origin 就可以使用 --allow-unrelated-histories 告诉 git 自己确定

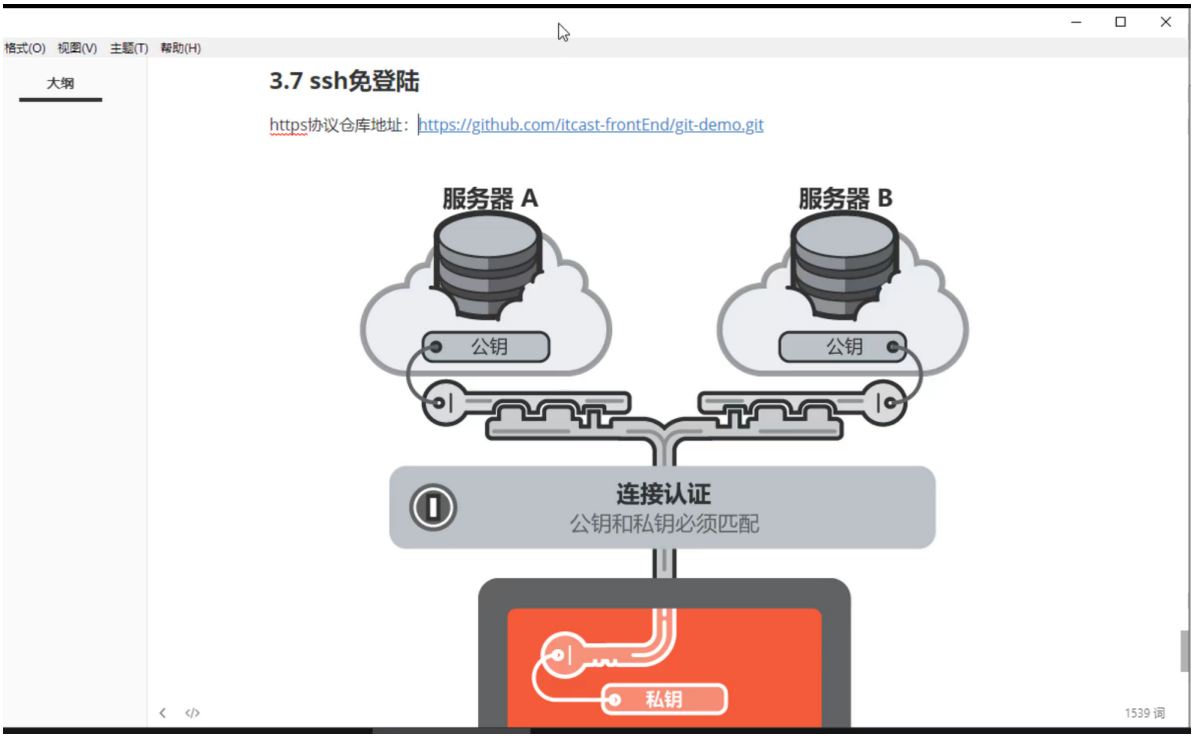
遇到无法提交的问题，一般先pull 也就是使用 git pull origin master 这里的 origin 就是仓库，而 master 就是需要上传的分支，因为两个仓库不同，发现 git 输出 refusing to merge unrelated histories 无法 pull 内容

————————————————————

改变文件夹

cd (文件夹名)

解决冲突



3.4 拉取操作

3.4.1 克隆仓库

克隆远端数据仓库到本地：`git clone 仓库地址`

3.4.2 拉取远程仓库中最新的版本

拉取远程仓库中最新的版本：`git pull 远程仓库地址 分支名称`

3.5 解决冲突

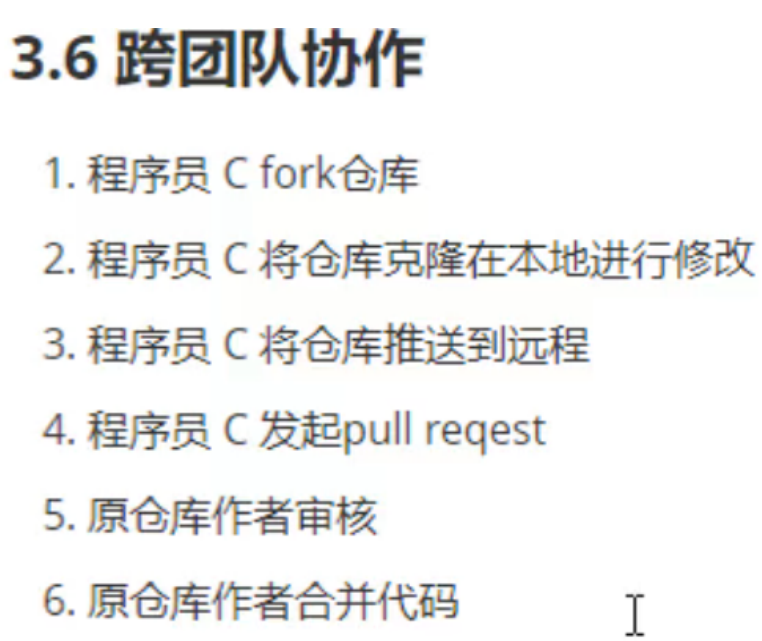
在多人同时开发一个项目时，如果两个人修改了同一个文件的同一个地方，就会发生冲突。冲突需要人为解决。

3.8 GIT忽略清单

将不需要被git管理的文件名字添加到此文件中，在执行git命令的时候，git就会忽略这些文件。

git忽略清单文件名称：.git

将工作目录中的文件全部添加到暂存区：`git add .`



生成秘钥

`ssh-keygen`

忽略清单

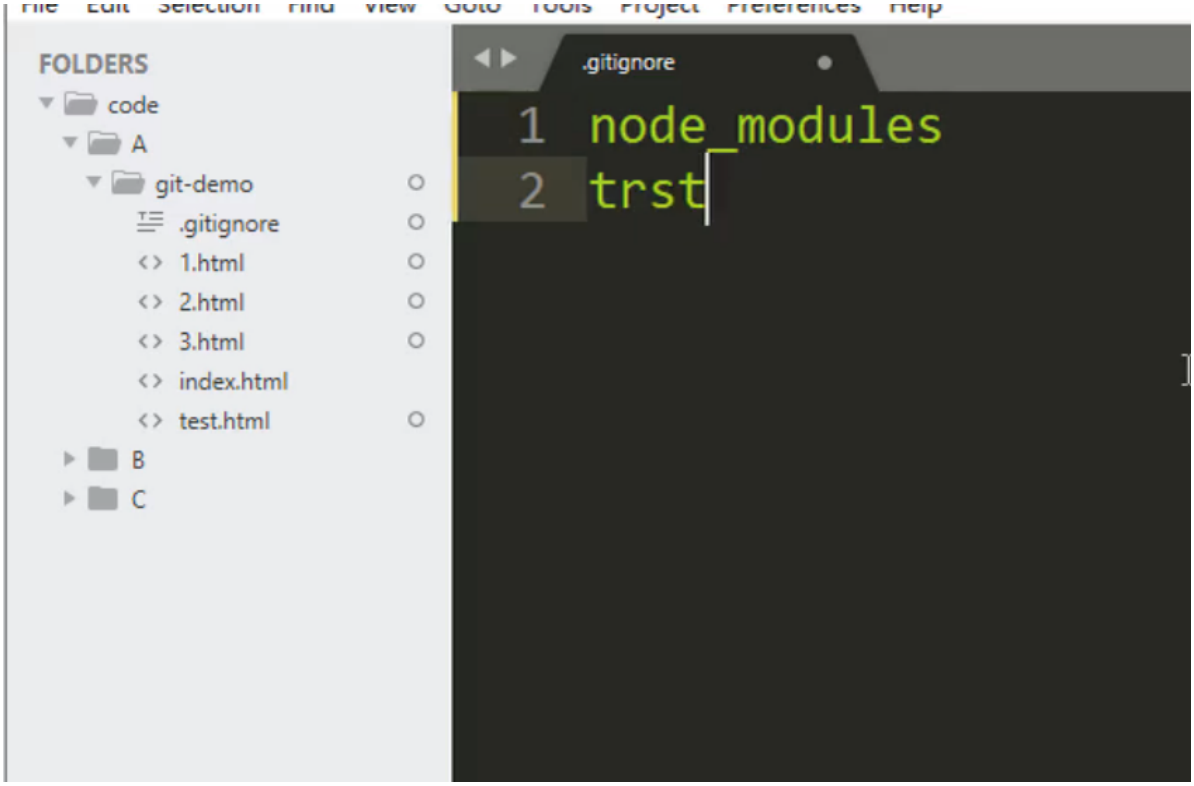
生成一个.gitignore文件

要以点开头

node_modules

（不需要被管理的文件）

在里写要忽略的文件



###仓库说明

上传md文件会变成仓库说明

分支重命名

`**git branch -m oldName**`

远程分支重命名 (已经推送远程-假设本地分支和远程对应分支名称相同)
a. 重命名远程分支对应的本地分支

`git branch -m oldName newName`

b. 删除远程分支

`git push --delete origin oldName`

上传新命名的本地分支

`git push origin newName`

把修改后的本地分支与远程分支关联

`git branch --set-upstream-to origin/newName`
1
注意：如果本地分支已经关联了远程分支，需要先解除原先的关联关系：

`git branch --unset-upstream`

———————————————————