**git checkout 影响区域**

1.工作目录

2.暂存区

3.HEAD

**后悔药**

工作目录 -> 暂存区 -> 版本库

回滚工作目录: 本事是覆盖

工作目录:v1 工作目录:v2

暂存区:v1 git checkout HEAD -- filename

版本库:v1 checkout 改HEAD指向 改暂存区 改工作目录

HEAD:v1

git checkout -- filename

git checkout HEAD – filename

回滾暂存区

git reset filename

git reset HEAD filename

撤回提交(假撤回)

提交时写错注释了

提交时忘记将真正的修改加到暂存区

git commit -m "xx" --amend

--amend 相当于 git reset --soft HEAD~

**reset三部曲**

第一部： soft 只覆盖head head会指向前一个对象

第二部: mixed 覆盖head 暂存区

第三部: hard 覆盖head 暂存区 工作目录

版本号: x.y.z;

升z位 : 解决bug

升y位 : 新增功能 / 新增api

升x为 : 工具或者代码库有了一次质的改变;整个使用习惯都有可能发生改变

**npm常用命令**

npm root -g : 查看全局安装的目录(存放全局包的目录一般会自动配在环境变量中)

npm init -y : 生产项目的配置文件 package.json

npm info 包 : 查看包的相关信息

npm i 包 -g

npm i 包@x.y.z -g

npm uninstall 包 -g : 全局安装&全局卸载

npm i 包 --save (npm i 包) : 局部安装生产环境的依赖

npm i 包 --save-dev(npm i 包 -D) : 局部开发开发环境的依赖

全局安装

可以任意目录下执行

局部安装

统一项目的版本号

.npx

在安装node时;会自动安装npm 和 npx

npx可以让我们在使用前端工具时不需要额外全局多安装一份;它会自动去找本地的工具

npx eslint -v

npm脚本

package.json : scripts 本质上是一个对象

key(脚本名称) : val(对应的命令)

如果脚本名字叫start 那run可以省略

npm run key --> 执行到对应的命令;它也会自动去找本地的工具

eslint

eslint –init ：初始化

eslint ./src：根据既定规则检查js代码

eslint ./src –fix : 根据既定规则检查js代码

结合husky为eslint加一个强约束

仓库初始化后在装husky

1. 初始化仓库 git init
2. 安装husky：npm i husky
3. 配置package.json文件

{

  "husky"**:** {

    "hooks"**:** {

      "pre-commit"**:** "npm test",

      "pre-push"**:** "npm test",

      "..."**:** "..."

    }

  }

}

保存文件 提交