一、主观题

1. 你会使用 Git 吗？（以面试者的口吻回答，要详细）

Git 是一个分布式版本控制系统，能在工作中高效管理代码版本、协作开发和解决冲突。

核心能力：

- **日常操作**：
  使用 `git clone` 复制远程仓库，`git add` 和 `git commit` 提交更改，`git push/pull` 同步远程代码，以及 `git branch/merge` 管理分支。
- **高级功能**：
  用 `git stash` 暂存未完成的工作，用 `git rebase` 整理提交历史，并通过 `git reflog` 恢复误操作。
  - 例如，在团队协作中，创建 `feature` 分支开发新功能，完成后通过 PR 合并到 `main` 分支。
- **场景经验**：
  曾处理过代码冲突：手动编辑冲突文件 → `git add` 标记解决 → `git commit` 完成合并。我还配置过 `.gitignore` 忽略编译文件（如 `target/`），确保仓库整洁。

2. 列举工作中常用的几个 git 命令？（以面试者的口吻回答，要详细）

工作中最常用的 Git 命令包括：

1. `git status`：
   查看工作区和暂存区状态。例如修改文件后运行它，会显示 `modified: file.txt`，提示未提交的变更。
2. `git add -A` 或者 `git add .`：
   将所有变更添加到暂存区。若只添加特定文件，用 `git add file.txt`。

3. `git commit -m "message"`：
   提交暂存区的变更到本地仓库。例如 `git commit -m "修复登录逻辑"` 会生成一个带描述的新版本。
4. `git push origin main`：
   推送本地提交到远程仓库的 `main` 分支。首次推送需关联分支：`git push -u origin main`。
5. `git pull`：
   拉取远程最新代码并自动合并（等价于 `git fetch` + `git merge`）。若本地有未提交变更，需先用 `git stash` 暂存。
6. `git checkout -b feature`：
   创建并切换到新分支（如 `feature`）。完成后用 `git merge feature` 合并回主分支。

3. 简单介绍一下 Git 的工作流程（以面试者的口吻回答，要详细）

Git 的标准工作流程分为五步：

1. **克隆仓库**：
   用 `git clone 仓库地址` 下载远程仓库到本地。
2. **创建分支**：
   基于 `main` 分支新建开发分支：`git checkout -b dev`。
3. **开发与提交**：
   在工作区修改代码 → `git add -A` 添加到暂存区 → `git commit -m "描述"` 提交到本地仓库。
4. **同步远程**：
   定期 `git pull origin main` 拉取团队更新，避免冲突；开发完成后 `git push origin dev` 推送分支。
5. **合并与部署**：
   提交 PR/MR 请求将 `dev` 合并到 `main`，通过代码审核后部署到生产环境。

4. Git 是用什么语言编写的?

Git 主要使用 **C 语言**编写，部分脚本用 Shell 和 Perl 实现。C 语言的高效性使 Git 能快速处理大型项目（如 Linux 内核）。

5. 什么是 git stash? 什么是 git stash drop?

- `git stash`：

  临时保存工作区的未提交变更，方便切换分支或拉取代码。例如：

  ```
  git stash           # 保存变更
  git pull            # 拉取远程代码
  git stash pop     # 恢复变更
  ```

- `git stash drop`：
  删除指定的暂存条目。
  - 例如 `git stash drop stash@{0}` 删除第一次保存的内容。
  - 与 `git stash pop`（恢复后自动删除）不同，`drop` 只删除不恢复。

6. 我提交版本是写了版本描述，发现写的不好，想修改，怎么修改?

使用 `git commit --amend` 修改最近一次提交的描述：

```
git commit --amend -m "新描述"   # 直接替换描述
```

若需修改提交内容，先更新文件 → `git add -A` → 再运行`git commit --amend`

注意：若已推送到远程，需强制推送（`git push --force`），但可能影响团队协作，应谨慎使用。

7. 大公司的分支一般有哪些分支？ 这些分支的工作流程是什么？

大公司常用 **Git Flow** 分支模型：

- **分支类型**：
  - `main` / `master`：生产环境代码，仅存放稳定版本。
  - `develop`：集成分支，合并所有新功能。
  - `feature/*`：功能分支，从 `develop` 分出，开发完成后合并回 `develop`。
  - `release/*`：预发布分支，用于测试和修复 Bug，合并到 `main` 和 `develop`。
  - `hotfix/*`：热修复分支，从 `main` 分出，紧急修复生产环境问题。
- **工作流程**：
  1. 新功能在 `feature/login` 分支开发。
  2. 完成 → 提交 PR 合并到 `develop`。
  3. 测试通过 → 从 `develop` 创建 `release/v1.0` 分支。
  4. 预发布测试 → 合并到 `main` 并打标签（`git tag v1.0`）。
  5. 生产环境 Bug → 从 `main` 创建 `hotfix/issue1`，修复后合并回 `main` 和 `develop`。

8. 练习一下分支合并（没有冲突和有冲突）

- **无冲突合并**：

```
git checkout main            # 切换到目标分支
git merge feature-login    # 合并 feature-login 分支
git push origin main         # 推送合并结果
```

若两分支无重叠修改，Git 自动完成合并（Fast-forward）。

- **有冲突合并**：

1. 合并时报错：`CONFLICT (content): Merge conflict in file.txt`。
2. 打开冲突文件，手动解决（删除 `<<<<<<<` `=======` `>>>>>>>` 标记）。
3. 标记解决：

```
git add file.txt          # 添加解决后的文件
git commit -m "解决合并冲突"   # 完成合并
```

9. 练习一下抓取远程分支和推送指定的分支到远程

- 抓取远程分支：

```
git fetch origin              # 获取远程所有分支信息
git checkout -b dev origin/dev   # 创建本地分支并关联远程分支
```

- 推送指定分支：

```
git push origin dev   # 推送本地 dev 分支到远程 dev
```

10. 什么是 clone 克隆、branch 分支和 fork 复刻？区别是什么？

- **Clone（克隆）**：
  复制远程仓库到本地，如 `git clone https://gitee.com/project.git`，创建完整的本地副本。
- **Branch（分支）**：
  在仓库内创建独立开发线，如 `git branch feature`，用于隔离开发新功能。
- **Fork（复刻）**：
  在代码平台（如 Gitee）复制他人仓库到自己的账户，创建独立新仓库，用于贡献开源项目。

- **区别：**

操作	范围	目的
Clone	本地	获取代码到本地
Branch	仓库内	并行开发新功能
Fork	平台账户之间	独立修改并提交回原项目

11. 练习 git fetch 拉取，然后和 git pull 对比

- `git fetch`：

```
git fetch origin    # 仅下载远程更新到本地仓库，不修改工作区
git merge origin/main    # 手动合并到当前分支
```

  - 适用场景：需先检查远程变更再合并。
- `git pull`：

```
git pull origin main    # 自动下载并合并远程 main 分支
```

  - 等价于 `git fetch + git merge`，但可能直接引入冲突。
- 对比：
  - `fetch` 安全可控；`pull` 便捷但风险更高（尤其本地有未提交变更时）。

12. 练习 git rebase 变基合并，然后和 git merge 对比

- `git rebase`：

```
git checkout feature
git rebase main    # 将 feature 的提交移到 main 分支顶部
```

  - 结果：历史线性整洁（无合并提交）。

    - 风险：重写历史，需避免在公共分支使用。
- `git merge`：

```
git checkout main
git merge feature    # 创建合并提交
```

    - 结果：保留分支历史（有分叉和合并点）。
- 对比：

操作	历史记录	适用场景
`rebase`	线性、简洁	私有分支整理提交
`merge`	保留分支拓扑	公共分支合并

13. Python 的特点是什么?

Python 的核心特点包括：

1. **简洁易读**：语法接近英语，如 `if age > 18:`。
2. **解释型语言**：无需编译，直接运行（如 `python script.py`）。
3. **跨平台**：支持 Windows、Linux、macOS。
4. **丰富的库**：标准库（如 `os`、`json`）和第三方库（如 `numpy`、`requests`）。
5. **动态类型**：变量类型在运行时确定（如 `name = "Alice"`）。
6. **多范式**：支持面向对象、函数式编程。

14. python 单行和多行注释是什么?

单行注释：以 `#` 开头

```
# 单行注释
```

多行注释：用三引号 `'''` 或 `"""`

```
'''
多行注释
第二行
'''
```

15. 使用 python 写一个 年龄判断、如果年龄大于 18 输出成年、否则输出未成年

```
age = int(input("请输入年龄: "))
if age > 18:
      print("成年")
else:
      print("未成年")
```

16. python 有哪些数字类型?

Python 的数字类型包括:

1. `int` **(整数)** : 如 `5`, `-10`。
2. `float` **(浮点数)** : 如 `3.14`, `-0.5`。
3. `complex` **(复数)** : 如 `1+2j`。

17.python 字符串支持几种写法?

字符串支持多种写法:

1. **单引号**: `'Hello'`
2. **双引号**: `"World"`
3. **三引号（多行）**:

18. 下面代码输出了什么?

```
str='123456789'

print(str)
print(str[0:-1])
print(str[0])
print(str[2:5])
print(str[2:])
print(str[1:5:2])
print(str * 2)
print(str + '你好')
```

```
1  str='123456789'
2  print(str)           #123456789
3  print(str[0:-1])     #12345678
4  print(str[0])        #1
5  print(str[2:5])      #345
6  print(str[2:])       #3456789
7  print(str[1:5:2])    #24
8  print(str*2)         #123456789123456789
9  print(str+'你好')     #123456789你好
```

- `[:]` 是复制整个字符串
- `[::-1]` 是反转字符串（你之前例子中学过）
- `[start:end:step]` 中：
  - `start` 默认是开头（正向为0，反向为-1）
  - `end` 默认是结尾（正向为len，反向为开头）
  - `step` 为负数时，方向反转!

二、进阶题

1. python最基础：写一个猜单价的游戏，提示用户输入一个单价，然后告诉用户价钱高了或者小了，直到用户输入准确的单价，结束，最后统计用户猜了几步。

```
# 设置正确单价
price = 80
count = 0

print("欢迎参加猜单价游戏！")
print("--------------------")

while True:
    # 获取用户输入
    guess = float(input("请输入您猜测的单价: "))
    count += 1

    # 判断猜测结果
    if guess > price:
        print("猜高了，试试低一点的价钱")
    elif guess < price:
        print("猜低了，试试高一点的价钱")
    else:
        print("恭喜您！正确答案是80，您猜对了！")
        print("您总共猜了", count, "次")
        break

```

三、 明天默写和面试题

运维班

1. 你知道 OSI 七层网络模型吗?

OSI（开放系统互联）模型是网络通信的标准化框架，分为七层：

1. **物理层**：传输原始比特流（如网线、光纤）
2. **数据链路层**：节点间数据传输（MAC地址、交换机）
3. **网络层**：路由选择和寻址（IP协议、路由器）
4. **传输层**：端到端连接管理（TCP/UDP协议）
5. **会话层**：建立和维护会话
6. **表示层**：数据格式转换和加密
7. **应用层**：用户接口（HTTP/FTP/SMTP协议）

2. 请说下 TCP 协议和 UDP 协议的区别。

特性	TCP	UDP
连接方式	面向连接（三次握手）	无连接
可靠性	可靠传输（确认重传）	尽力交付（可能丢包）
顺序保障	保证数据顺序	不保证顺序
速度	较慢	较快
头部大小	20-60字节	8字节
适用场景	文件传输、网页浏览、邮件	视频流、在线游戏、DNS查询

3. 你知道哪些常用的端口号?

- **21**：FTP（文件传输）
- **22**：SSH（安全远程登录）
- **25**：SMTP（邮件发送）
- **53**：DNS（域名解析）
- **80**：HTTP（网页访问）
- **110**：POP3（邮件接收）
- **143**：IMAP（邮件同步）
- **443**：HTTPS（安全网页）
- **3306**：MySQL数据库
- **3389**：远程桌面协议(RDP)

4. 为什么我们不使用 root 账户?

1. **安全风险**：
   - 单点故障：root 泄露等于系统完全失守
   - 恶意命令：`rm -rf /` 等破坏性命令无阻拦
2. **最小权限原则**：
   - 普通用户需 `sudo` 提权执行特权操作
   - 操作审计（`/var/log/auth.log` 记录 sudo 操作）
3. **防误操作**：
   - 避免意外修改系统关键文件（如 `/etc`）
   - 防止误删系统文件导致宕机
4. **安全纵深防御**：
   - 攻击者突破普通账户仍需提权
   - 提供二次防护机会（如监控异常 sudo 请求）

答案在：http://47.110.66.96/helper 运维部分

鸿蒙班

1. vue 的生命周期有哪些?

2. vue 的组件有哪些通讯?

3. vue 的 key 作用?

4. vue 的常见指令有哪些?