# Git 工作流规范

## 1. 名词定义

- PR: product request, 产需、产品需求，负责和客户明确需求，给出产品原型，输出需求文档
- PM: project manager, 产品管理/产品经理，负责管理项目运作，以及人力/时间等资源的协调
- PO: project owner, 产品拥有人，负责协调项目代码，包括代码审查(review)，代码分支管理等
- Dev: developer, 开发者，负责完成自己的开发任务(task)，并完成白盒测试（主要单元测试，其次集成测试）
- DevTeam: develop team, 开发团队，负责整个项目的开发工作
- Test: tester, 测试人员，负责对开发的功能进行测试，包括需求测试、功能测试（黑盒测试）、集成测试等
- Ops: operator, 运维人员，负责对各个运行环境的搭建和运维工作，包括环境搭建、日常运维、运维监控等
- Cfg: configurator, 配置人员，负责对代码仓库基线(baseline)的控制，以及项目文档的归档管理等
- request: 需求，指一个完整的功能或者模块，或者一系列相关联的功能
- task: 任务，是将需求拆分为多个独立编码任务的单元，分配给具体的开发者(developer)
- archive: 制品，是代码的输出结果，是可运行的程序，交付的产品

## 2. 敏捷开发(Scrum)项目管理体系

- Scrum 使用【故事/任务】+【阶段需求】来规范开发流程
- 【阶段需求】与【需求号】挂钩
- 上线以【需求号】为准
- 因此，结合实际的Scrum流程
- 串接Git工作流，才能实现最佳适配
- 下面，来制定Git工作流程
- 同时，敏捷开发，是当下主流的项目管理方式
- 可能会根据公司的实际情况，有一定的变体

### 2.1. 敏捷流程

- 产需(PR/PM)每周根据下阶段上线需求(request)拆分为任务(task)
- 在例会上，产需(PR/PM)给开发团队(DevTeam)讲解上线需求(request)，并将任务(task)分配给开发者(developer)
- 开发者(developer)进行编码、调试工作，完成自己的任务(task)，并将自己任务结果提交到测试环境(test)，并提交给测试人员(test)进行测试
- 测试人员(test)进行测试提出问题(issue)后反馈给开发者(developer)，开发者(developer)进行修复，测试人员(test)进行回归测试
- 直到，功能满足需求(request)描述，结束这个开发与测试的流程
- 在整个需求(request)完成测试之后，进入上线发布环节
- 项目管理(PM)编写上线材料，提交给配置人员(Cfg)，完成需求确认
- 配置人员对已确认的需求相关的代码分支合并到主分支(release/master)，完成基线(baseline)提升
- 配置人员(Cfg)通知运维人员(Ops)打包主分支(release/master)，生成制品程序(archive)
- 运维人员(Ops)将制品(archive)部署到生产环境，完成产品升级
- 测试人员(Test)对升级的产品进行验证测试，当验证出现问题时，需要通知开发团队(DevTeam)进行紧急修复(hotfix)
- 开发团队(DevTeam)完成紧急修复(hotfix)验证通过之后，整个需求完成上线发布
- 到此流程就完毕了

## 3. Git 工作流

- 使用准则
  - 单向流动原则
    - 保持分支提交的单向流动，保持分支关系之间是有向无环图（DAG）关系
    - 特别是严格控制分支之间的相互合并
  - 最小化原则
    - 新增/修改的特性分支，应该只包含此特性所涉及的变更
    - 不要带入其他未加入基线的特性
  - 基线唯一原则
    - 永远保持只有唯一的一个基线分支（主分支）

        - 任何新的特性或者修复，都应该从基线分支拉取
        - 永远都应该相信基线，任何时候都可以从基线分支拉取分支
        - 任何时候都允许将基线分支合并到其他分支
    - 先拉后推原则
        - 在想要进行push推送操作之前
        - 先进行pull拉去操作，尽可能的将merge和conflict在push之前完成
    - 单一职责原则
        - 对分支的修改，应该只在自己分支上进行，而不应该跨分支修改
        - 特别是revert,rollback,rebase操作
        - 最好只在自己分支上操作，不要merge到其他分支之后，在其他分支上进行这些操作
        - 因为这样可能引起混乱
    - 谨慎原则
        - 再使用一些git操作时，请谨慎操作
        - 比如 force push, rebase 操作
        - 这可能会造成分支混乱

### 3.1. 分支示例

- release
- test
- request
    - REQ26634-1
    - REQ16455-1
    - REQ17788-1
    - REQ17788-2
- feature
    - sms-notice
    - dashboard
    - scheduler
    - home-bigscreen
    - g-35548
- hotfix
    - 240808
- publish
    - 240808

### 3.2. 主要分支

- release: 别名 master/product， 主分支/基线分支
    - 基线分支，任何分支都应该从基线分支建立
    - 只允许request分支合并到此分支
    - 不允许删除此分支
    - 不允许在此分支提交代码
    - 允许任何时间将此分支合并到其他分支
- test: 测试分支
    - 测试分支，任何已开发的分支都允许合并到此分支
    - 只允许feature/hotfix合并到此分支
    - 不允许删除此分支
    - 不允许在此分支提交代码
    - 不允许将此分支合并到其他分支
- request: 需求分支/阶段分支
    - 需求分支，每个阶段明确上线的内容
    - 只允许publish分支合并到此分支
    - 不建议删除此类分支，但需要定期清楚老旧分支
        - 此类分支数量需要保留一定的数量
        - 可根据自己团队情况，保留3个以上10个以内的近期需求

    - 不允许在此分支提交代码
      - 除非在还未上线时，通过项目团队人员确认之后
      - 可允许少量变更直接在此分支提交变更
    - 只允许此分支合并到release/test分支
      - 原则上，此分支涉及的内容，早已经在test分支经过了测试
      - 所以，通常来说，只允许合并到release分支
      - 不需要合并到test分支
      - 命名规则：request/需求号-阶段号
        - 例如：request/REQ26634-1
        - 表示需求26634的第1个阶段
      - 分支创建
        - 此分支，在上线申请填写完毕之后，由PM/PO给出需求号（例如REQ26634-1）
        - 并指定任意开发人员(或开发组长)，从release分支创建此需求分支 request/REQ26634-1
        - 并将指定的publish预发布分支，合并到此需求分支，完成分支内容合并
        - 最后，交由配置人员，打包编译发布
        - 由配置人员合并此分支到release分支
        - 完成上线
- feature: 别名 develop， 特性分支/开发分支
  - 特性分支，各个开发根据自己的开发任务/需求故事完成编码
  - 严禁将test/request/publish分支合并到此分支
    - test分支包含了不属于你这个feature分支应该包含的代码
    - 因此不能合并
    - request和publish分支的原因也一样
    - feature分支，应该只能包含release/hotfix和你自己feature的代码
    - 不要包含其他任何代码
    - 只允许release/hotfix分支合并到此分支，严禁feature分支之间相互合并
      - 到出现多个feature分支之间需要协作时
      - 应该将这几个feature分支合并为一个新的feature分支
      - 原来各自负责feature的开发人员，共同维护新的这个feature分支
      - 并删除原来那些独立的feature分支
    - 建议删除此类分支，在完成合并到release分支之后
      - 在完成合并到release分支之后
      - 表示本次feature已经完成上线
      - 不应该再继续在此分支上继续修改
      - 而是应该从release分支建立新的feature分支继续开发
      - 保证与release的基线(baseline)一致
    - 命名规则：feature/自定义名称
      - 例如：feature/home-bigscreen
      - 表示首页大屏的需求开发
      - 例如：feature/g-35548
      - 表示进行35548这个故事编号的需求开发
    - 分支创建
      - 各个开发从release分支创建各自的需求分支 feature/home-bigscreen
    - 需求冲突
      - 如果多个开发人员，开发内容涉及到同样的内容
      - 则应该将这冲突的多个开发人员的分支合并为一个新分支
      - 多个开发人员同时维护这一个分支的代码
      - 在进行任意的push操作之前
      - 必须先进行update操作，并解决可能的conflict冲突
      - 在本地完成merge之后，再进行push操作
- hotfix: 热修复分支/线上问题修复分支
  - 热修复分支，通常用于再上线完毕后，或临时处理线上问题时，进行代码的调整
  - 只允许从release分支创建此分支
  - 如果release分支还未完成最后一次的request分支的合并时

        - 则使用最后一次request分支代替release分支（不建议）
    - 允许将此分支合并到其他任意分支
        - 原则上来说，hotfix分支已经在线上允许
        - 属于release分支的一部分，属于基线内容
        - 所以，允许合并到任意分支
        - 而hotfix分支存在的意义就是在下一次提升release基线的时候
        - 加入之前的hotfix内容，而不是频繁的变更release基线（或许其他各种原因，导致提升基线比较麻烦）
        - 建议删除历史分支
            - 在hotfix分支合并到release分支之后
            - 表示热修复工作已经完成
            - 不在需要此分支
            - 因此建议删除此分支
        - 命名规则：hotfix/日期
            - 例如：hotfix/240808
            - 表示24年8月8号进行的热修复内容
            - 当出现需要多人一起进行hotfix时
            - 建议由其中一人，先创建hotfix分支
            - 其他人员在此分支上，共同提交修复内容
            - 或者各自创建自己的hotfix分支
                - 命名规则：hotfix/日期-姓名
                - 例如：hotfix/240808-miler
        - 分支创建
            - 如果，已经完成request分支合并到release分支后进行的热修复
                - 则从release分支创建热修复分支 hotfix/240808
            - 如果，还未完成request分支合并到release分支进行的热修复（不建议）
                - 则从request分支创建热修复分支 hotfix/240808
- publish: 别名 preview/pre/uat， 预发布分支
    - 预发布分支，由于request需求分支，需要填写上线申请之后或其他原因，导致没有需求号，无法创建分支
    - 则可以使用预发布分支，进行对request分支内容的预先合并，request分支的内容保持和publish分支一致
    - 此分支只允许从release分支创建
    - 只允许release/feature/hotfix分支合并到此分支
    - 只允许将此分支合并到request分支
        - 此分支，只能合并到request分支
        - 不允许合并到其他分支，也不允许从此分支创建新分支
        - feature分支应该时PM/PO指定要上线的需求涉及到的内容的分支
        - 建议删除历史分支
            - 当完成合并到release之后
            - 表示已经完成上线
            - 预发布分支已经失去意义
            - 可以删除历史分支
            - 保留最后1~2个分支，用以表示上次上线时间即可
            - 用来和hotfix分支进行对照，以确定哪些hotfix分支是有意义的
        - 命名规则：publish/日期
            - 例如：publish/240808
            - 表示在24年8月8号（或者计划上线日期）创建的预发布分支，下次上线的内容就是最后一个预发布分支的内容
            - 在上线时，request分支直接从最后一个publish分支创建
- tmp: 别名 temp，临时分支
    - 临时分支，用于暂存中间结果
    - 使用再进行merge发生冲突时，或者其他时候进行冲突的解决或者其他操作
    - 此分支，应该用完之后立即删除

### 3.3.一般流程

- release - feature - test - publish - request - release
- 各个开发者(developer)根据自己的任务(task)从release分支创建自己的特性分支(feature)

- 开发者(developer)完成任务(task)的开发工作之后，自行合并(或PO审查并合并)到test分支
- 本阶段需求完成后，PM/PO(也可以是指定开发者，比如开发组长)从release分支创建预上线分支publish
- 各个涉及到本次上线需求任务(task)的开发者(developer)将自己的feature分支合并到publish分支
- PO(或者指定的开发组长)将上阶段的所有hotfix合并到publish分支
- 合并期间，如果发生代码冲突，则由feature的开发者(developer)与其他成员确认解决冲突后合并
- PM进行上线材料申请，上线时间到达之后，PO从release分支创建request分支，并将publish分支合并到request分支
- 配置人员(Cfg)将request分支合并到release分支，完成基线(baseline)提升
- 运维人员(Ops)从release分支打包输出制品(archive)进行部署到生产环境
- 测试人员(Test)进行验证测试，如果发现验证问题(issue)
- PR/PM确认是必要紧急修复的问题的话，开发团队(DevTeam)进行热修复
- 开发者(developer)从release分支创建hotfix分支，修复问题并提交，注意同时合并到test分支
- 验证通过后完成上线发布流程，本轮流程结束

### 3.4. 案例讲解

- 上次上线热修复分支
  - hotfix/240808
- 本次计划上线日期
  - 240822
- 计划上线需求号
  - REQ26153-1
- 分配的开发任务
  - active-report 活跃报表
  - fee-report 费用报表
  - flow-upgrade 流程升级
- 则分支应该如下
  - release
  - test
  - hotfix
    - 240808
  - feature
    - active-report
    - fee-report
    - flow-upgrade
  - publish
    - 240822
  - request
    - REQ26153-1
- 则合并规则如下
  - 从release分支分别创建feature/active-report,feature/fee-report,feature/flow-upgrade分支进行开发
  - 如果指定的feature依赖hotfix的内容，则允许将hotfix分支合并到feature分支
  - feature/active-report,feature/fee-report,feature/flow-upgrade合并到test完成测试
  - 从release分支创建publish/240822预发布分支
  - 将feature/active-report,feature/fee-report,feature/flow-upgrade分支合并到publish/240822分支
  - 将hotfix/240808分支合并到publish/240822分支
  - 到这里，预发布分支已经完成合并与解决冲突
  - 从release分支创建request/REQ26153-1需求分支
  - 将publish/240822分支合并到request/REQ26153-1完成需求分支
  - 将request/REQ26153-1分支合并到release分支完成上线基线提升

### 3.5. Git使用约束

- 在进行push之前，先进行pull拉去更新，解决可能得conflict冲突
- 谨慎使用rebase
- 谨慎使用force push
- 注意使用cherry-pick

- 建议统一commit时进行代码格式化
- 建议统一使用commit message的格式
- 格式定义如下

```
提交类型(业务范围)：修改描述

修改详情
```

- 提交类型定义
  - feat/feature: 新功能，功能调整
  - fix/fixed: 问题修复，调整
  - doc/docs: 文档或者注释的变更
  - style: 代码格式调整，格式化，去除无用导入类等
  - refactor: 功能代码的重构，类名、包名、方法名、变量名等重命名等
  - perf: 性能优化调整
  - test: 单元测试代码
  - revert: 撤回提交
- 举例

```
feat(费用报表)：添加统计费用报表逻辑

添加定时报表生成
添加报表查询接口
添加报表导出功能
```

- 在IDEA中，可以使用插件(***Git Commit Template Check***)
  - 或者[***Git Commit Message Helper***]插件
- 这样就可以统一开发团队的提交信息

# 4. 场景说明

## 4.1. 收到新开发任务

- 开发人员(developer)在收到新开发任务(task)时
- 应该从release分支创建一个自己任务的feature分支进行开发
- 举例说明
- 接到的开发任务(task)叫做：活跃报表
- 那么，可以起个名字：active-report
- 那么，从release分支创建feature/active-report分支
- 然后再feature/active-report分支进行开发
- 在经过自测后，提交到测试分支，将feature/active-report分支合并到test分支
- 在测试人员(Test)提出问题(issue)之后，继续在feature/active-report分支进行修改
- 修改完成之后，再将feature/active-report分支合并到test分支进行测试
- 依次循环，直到消除所有可见的问题(issue)，满足需求为止

## 4.2. 线上出现问题了，需要及时修复

- 开发人员(developer)在收到线上问题(issue)时
- 原来负责此相关功能(feature)的开发人员/被指定的开发人员
- 应该从release分支创建一个hotfix分支进行问题修复
  - 如果此前已经存在了未合并到release分支的hotfix分支
  - 则可以允许直接使用原来的hotfix分支
- 举例说明
- 假如今天日期是20240808，发现了线上问题
- 则责任开发人员从release创建hotfix/240808分支
  - 如果发现，上一次的发布分支是publish/240802

    - 表示上次上线是在20240802进行的
    - 此时，在此日期之后，已经存在了一个hotfix分支hotfix/240803
    - 则可以直接在hotfix/240803分支上进行修复
- 并在hotfix/240808进行问题修复
- 问题修复后，提交代码，并将hotfix/240808分支合并到test分支给测试人员测试验证
- 验证通过之后，问题修复完成
- 根据各个公司的实际情况
    - 可能选择使用arthus等热修复工具直接修复
    - 也可能需要本地打包，这里说本地打包的解决方式
        - 本地打包，则先明确需要包含的代码有哪些
        - 其实就是release分支的代码，以及从上次上线以来的所有hotfix分支的内容
        - 则从release分支创建tmp/240808分支
        - 将hotfix/240808分支和hotfix/240803分支合并到tmp/240808分支
        - 将tmp/240808分支打包进行上线部署
        - 部署验证完毕之后
        - 删除tmp/240808分支，完成线上问题修复

### 4.3. 我的feature需要和同事的feature协作

- 开发人员在开发时，难免会发生两个或者多个开发任务(task)修改需要进行协作的情况
- 当这种情况发生时，原本各自维护的feature分支，就需要进行合并为一个新的feature分支
- 然后，这些需要协作的开发人员共同维护这个新的feature分支
- 举例说明
- 现有分支如下
    - feature
        - user-manage
        - role-manage
        - dept-manage
- 对于开发feature/user-manage分支的开发人员来说
- 他需要用到role-manage和dept-manage的内容
    - 某种程度上，这时候可能需要向开发组长等报告进行协调其他开发人员配合
    - 根据实际情况决定
- 这时候，就可以从release分支创建新的feature分支，比如feature/rbac-manage
- 然后将user-manage,role-manage,dept-manege合并到rbac-manage分支
- 随后删除user-manage,role-manage,dept-manage分支
- 原来的开发人员，共同在rbac-manage分支进行协作开发
    - 这个时候，请一定要记住先拉后推原则

### 4.4. merge时发生conflict冲突

- 当进行pull更新，或者merge操作时
- 可能会发生conflict冲突
- 当冲突发生时，冲突所涉及到的代码变更
- 需要代码提交涉及的开发人员共同解决冲突
    - 一般来说，各自负责自己的feature进行开发
    - 并不知道其他开发人员的意图的情况下
    - 不要进行盲目的冲突解决
    - 而是共同决定冲突的解决方式
    - 除非，你明确的知道为什么冲突，以及应该怎么解决
- 注意事项
    - 冲突合并时，需要认真观察合并后的结果
    - 有时候在代码合并之后，可能产生多余的符号，重复定义的符号等
    - 因此，建议认真观察合并后的结果
    - 并且，在合并完成之后，完成一次本地编译
    - 确认合并的结果没有语法错误

- 合并准则
  - 原则上，任何分支都不应该和release分支发生冲突
  - 否则，那就是git使用不规范导致的
  - 一般来说，提交可能和test分支冲突
  - 这种情况就是多个feature修改了同样的地方导致的
  - 这个时候，两种方式
    - 能直接修改提交test分支的情况
      - 这可以拉取test分支到本地，在本地完成合并后推送到test分支
    - 不能直接修改稿提交test分支的情况
      - 那就从test分支创建一个临时分支
      - 将自己的分支合并到这个临时分支，完成冲突解决
      - 再将临时分支合并到test分支
      - 最后删除临时分支

### 4.5. 项目需要上线

- 当项目通知要上线时或者明确了要上线的内容做上线准备时
- 会给出需要上线的内容
- 开发组长或者指定的责任开发人员，从release分支创建publish/240808预发布分支
- 同时，将上次上线以后产生的所有hotfix分支合并到publish/240808分支
- 涉及到上线内容的开发人员
- 将自己负责的feature分支合并到publish/240808分支
- 开发组长或者指定的责任开发人员，更新publish分支到本地
- 检查publish分支与release分支差异，确定分支内容符合上线内容
- 以及确认有无编译问题
- 确认无误后，涉及到的feature分支与hotfix分支，理论上不再发生变更
- 但根据实际情况，可以发生变更，变更后重新合并到publish分支

### 4.6. 项目正式上线

- 当项目正式上线时间到达之后
- 项目组长或者指定的责任开发人员，从release分支创建request/REQ26153-1
- 并且，将publish/240808分支合并到request/REQ26153-1
  - 可以将hotfix分支再合并到request分支
  - 不过一般来说，publish分支就已经确认了上线内容
  - 不需要再合并一遍
- 项目经理通知配置人员进行基线提升
- 配置人员将request/REQ26153-1分支合并到release分支，完成基线机身
- 运维人员从release分支打包出上线的制品(Archive)进行上线部署
- 接下来，测试人员完成线上验证测试
- 开发人员配合完成可能产生的线上问题进行hotfix
- 最后，上线完成

## 5. IDEA/Jetbrains 家族的集成开发环境中，使用GIT

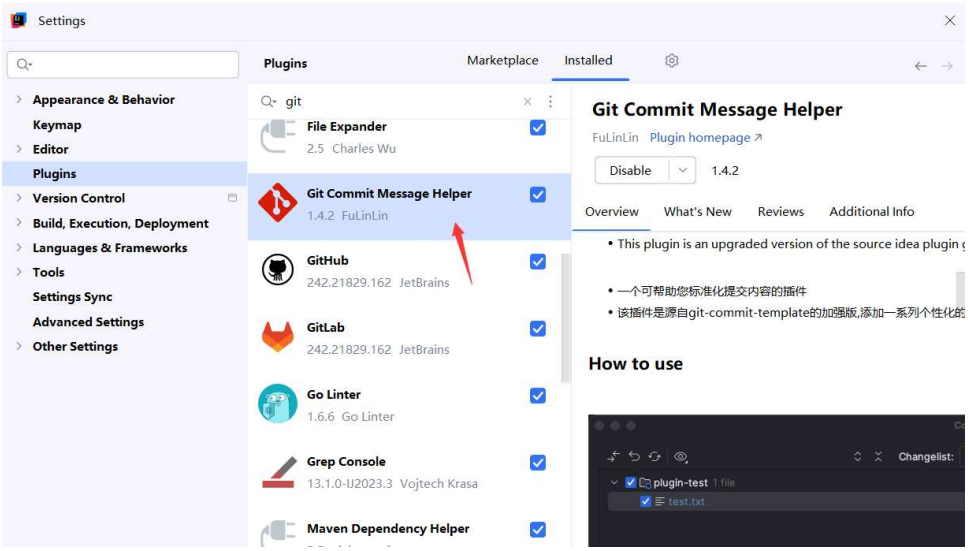
- 各个版本的IDEA中，GIT的位置略有差异
- 但是主要使用的菜单名称都是一样的
- 这里只是简要的介绍一些常用的功能

### 5.0. 提交代码

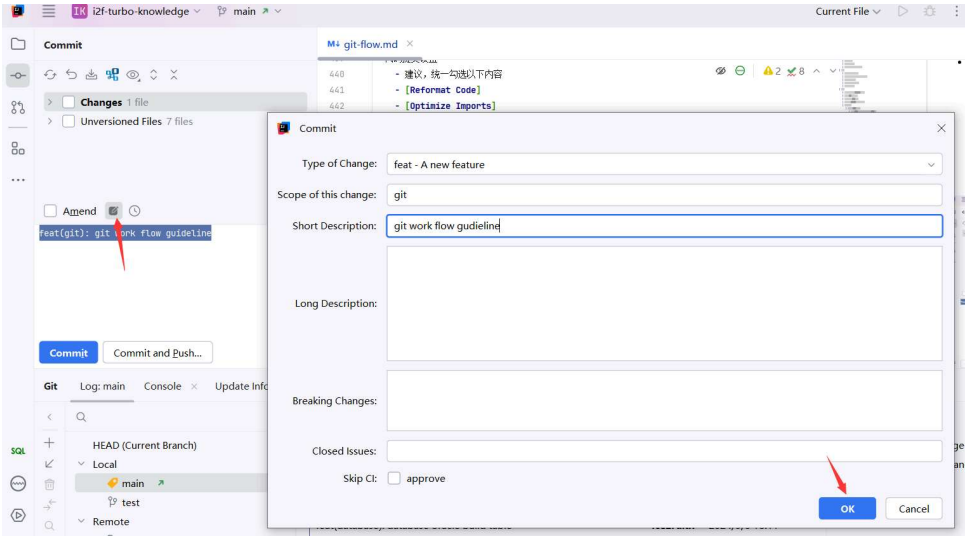
- 建议安装IDEA插件[***Git Commit Template Check***]来规范commit message格式
  - 或者[***Git Commit Message Helper***]插件
- 代码提交设置
  - 建议，统一勾选以下内容

  - [Reformat Code]
  - [Optimize Imports]
  - 这个只在适用于格式化的场景下勾选
  - 如果提交只涉及到一些无关乎代码的提交
  - 可以不勾选

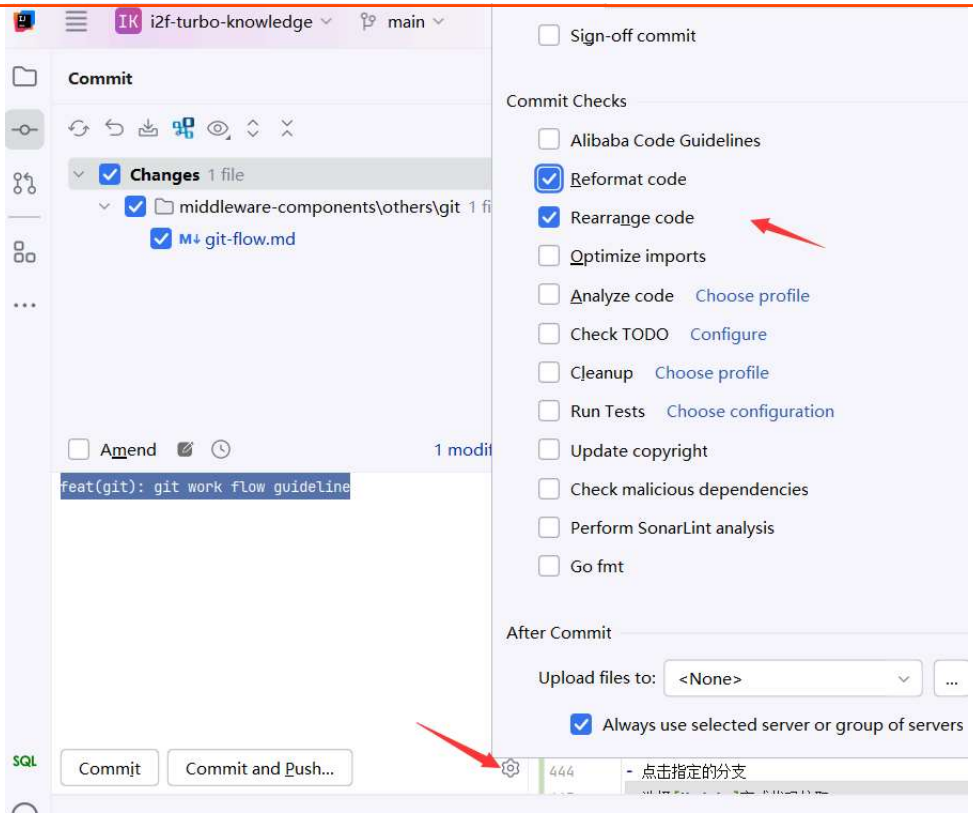
- 插件 images/git-commit-plugin.png




- 提交消息 images/git-commit-message.png

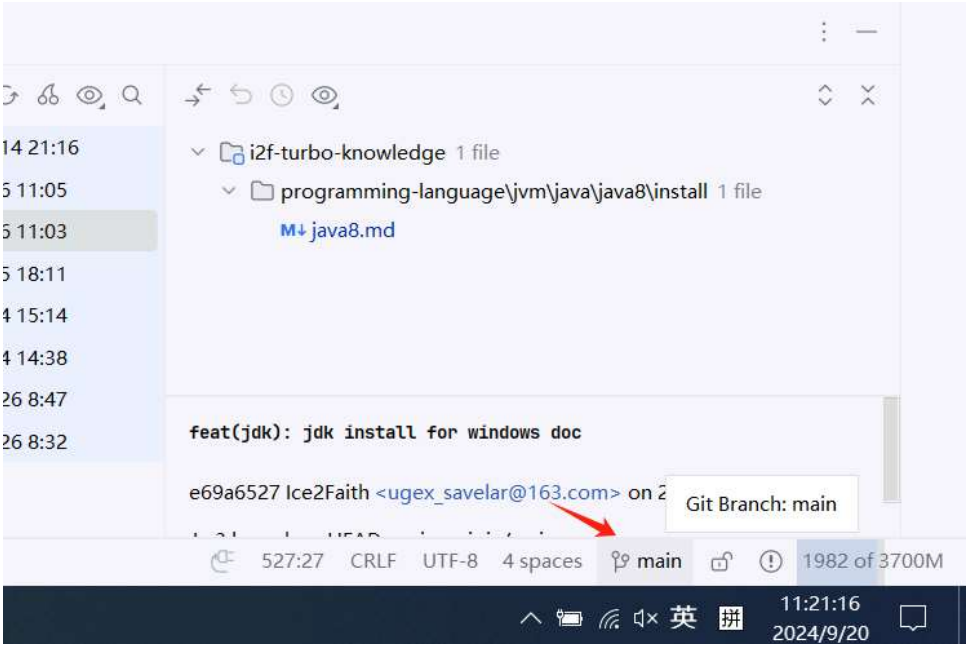



- 提交设置 images/git-commit-setting.png

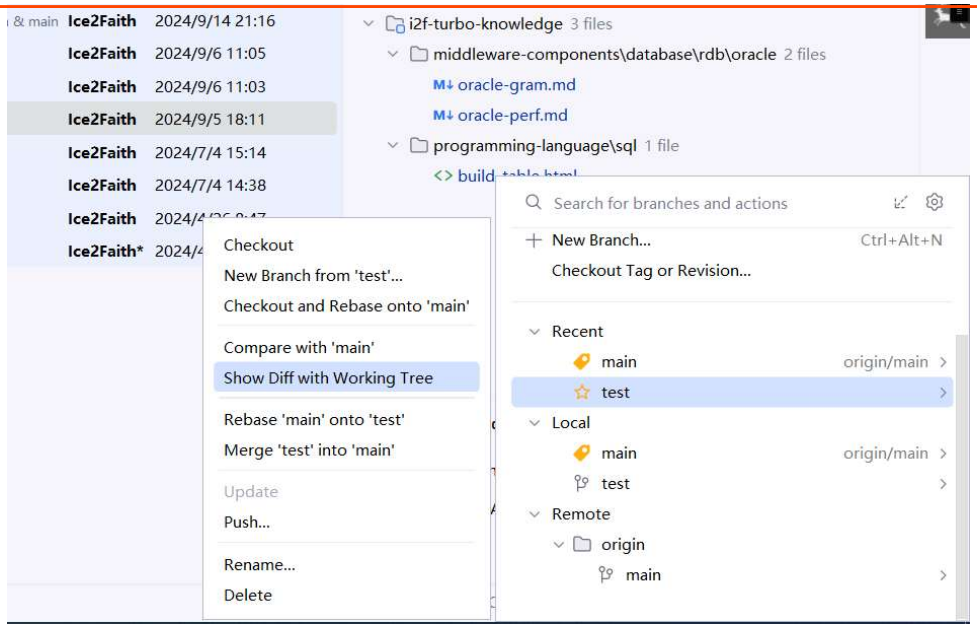



### 5.1. 检出指定分支

- 右下角，点开GIT状态栏
- 点击指定的分支
- 选择[Check]完成检出
- 这时候，会自动切换到新检出的分支
- GIT状态栏 images/git-status.png

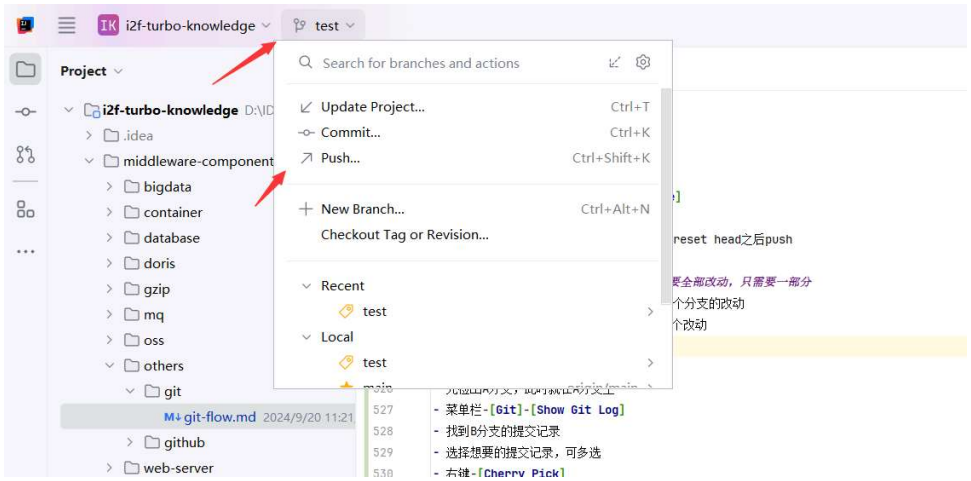



- GIT状态栏菜单 images/git-status-menu.png




## 5.2. 拉取代码

- 右下角，点开GIT状态栏
- 点击指定的分支
- 选择[Update]完成代码拉取
- Git快捷操作 images/git-shortcut.png




## 5.3. 推送代码

- 右下角，点开GIT状态栏
- 点击指定的分支
- 选择[Push]完成代码拉取

## 5.4. 从指定分支创建新分支

- 右下角，点开GIT状态栏
- 点击指定分支
- 选择[New Branch From 'xxx']完成新分支创建
  - 注意，这样创建的分支是再你本地
  - 并没有再远程仓库
  - 可以使用push推送到远程仓库
  - 则远程仓库也会存在创建的这个分支
- 这时候，会自动切换到新创建的分支

### 5.5. 分支合并

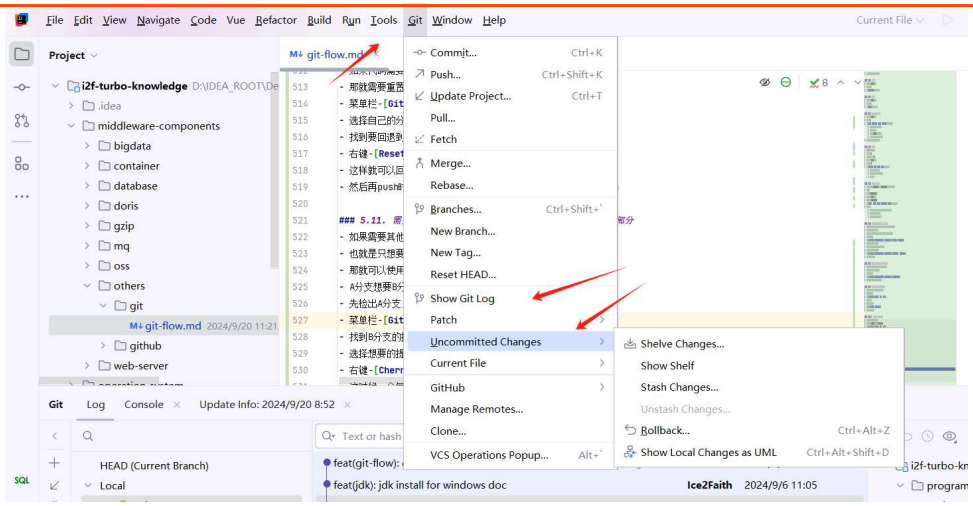
- 分支A要合并到分支B
- 则先检出分支B
- 这时候，当前就是分支B
- 右下角，点开GIT状态栏
- 点击指定分支A
- 选择[Merge 'A' into 'B']完成分支合并
  - 合并过程中，可能存在冲突
  - 再弹出的对话框中，打开冲突的文件
  - 进行冲突解决

### 5.6. 代码差异到底有哪些

- 如果我想要看我的feature分支和release分支的代码与那些差异
- 则这样看
- 先检出我的feature分支
- 右下角，点开GIT状态栏
- 点击指定分支release
- 选择[Show Diff with Working Tree]完成新分支创建
- 有什么用?
  - 代码要上线了，但是需要检查上线内容是不是指定的上线内容
  - 有没有包含不需要上线的代码
  - 或者，想要看看和其他分支的差异

### 5.7. 代码还没提交，但是要切换到其他分支修改

- 如果代码还没提交，又不具备提交的条件（比如代码写一半，连编译都过不了）
- 但是又要切换到其他分支修改代码
- 直接检出分支切换的话，两个分支代码交叉，分辨不了或者提交带来麻烦
- 那就需要暂存代码
- 菜单栏-[Git]-[Uncommited Changes]-[Stash Changes]
- 保存为一个暂存区，需要给个名字
- 那么，其他分支修改完了，我需要回来继续写代码
- 就需要从暂存区取出代码继续开发
- 菜单栏-[Git]-[Uncommited Changes]-[Unstash Changes]
- 选择一个暂存区恢复即可
- GIT菜单栏 images/git-menu.png

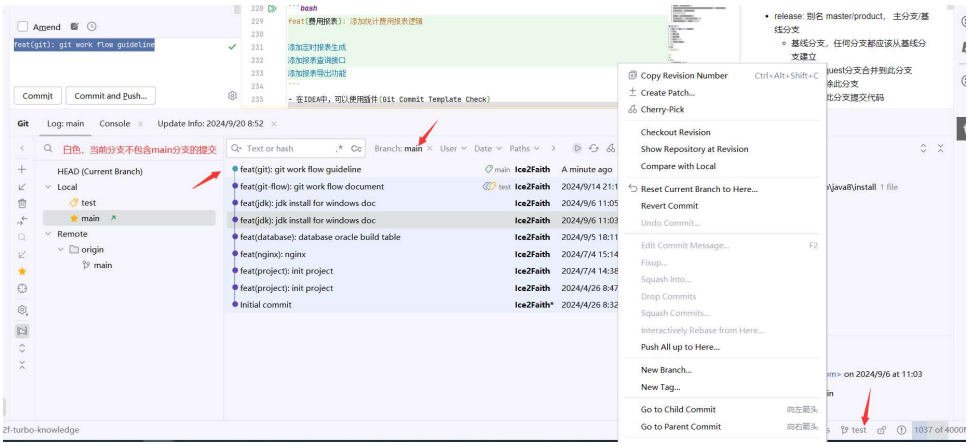



## 5.8. 代码还没提交，但是不想要提交刚才的修改

- 如果代码还没提交，刚才改动了内容，这些改动又不需要
- 比如，发现改错了，或者需求变了等
- 那就需要回滚代码
- 菜单栏-[Git]-[Uncommited Changes]-[Rollback]
- 可以勾选你需要回滚的文件进行撤销修改稿

## 5.9. 代码提交了，但是需要撤销这个修改

- 如果代码已经提交了，但是又想要撤销某个变更
- 那就需要撤销变更
- 菜单栏-[Git]-[Show Git Log]
- 选择自己的分支
- 找到自己的提交日志
- 右键-[Revert Commit]
- 这样就可以撤销修改了
- GIT日志 images/git-log.png




## 5.10. 代码需要回退到之前的版本

- 如果代码需要回退到以前的版本
- 那就需要重置HEAD，也就是rebase
- 菜单栏-[Git]-[Show Git Log]
- 选择自己的分支
- 找到要回退到的提交日志

- 右键-[Reset Current Branch to Here]
- 这样就可以回退版本了
- 然后再push时进行force push即可，或者reset head之后push

### 5.11. 需要其他分的的改动，但是不需要全部改动，只需要一部分

- 如果需要其他分支的改动，但是又不需要整个分支的改动
- 也就是只想要一部分改动，而不是merge整个改动
- 那就可以使用Cherry Pick选取提交
- A分支想要B分支的一些提交
- 先检出A分支，此时就在A分支上
- 菜单栏-[Git]-[Show Git Log]
- 找到B分支的提交记录
- 选择想要的提交记录，可多选
- 右键-[Cherry Pick]
- 这时候，会每个提交记录都会弹出提示，提示你提交
- 你自己提交，合并，解决冲突完成Cherry Pick即可