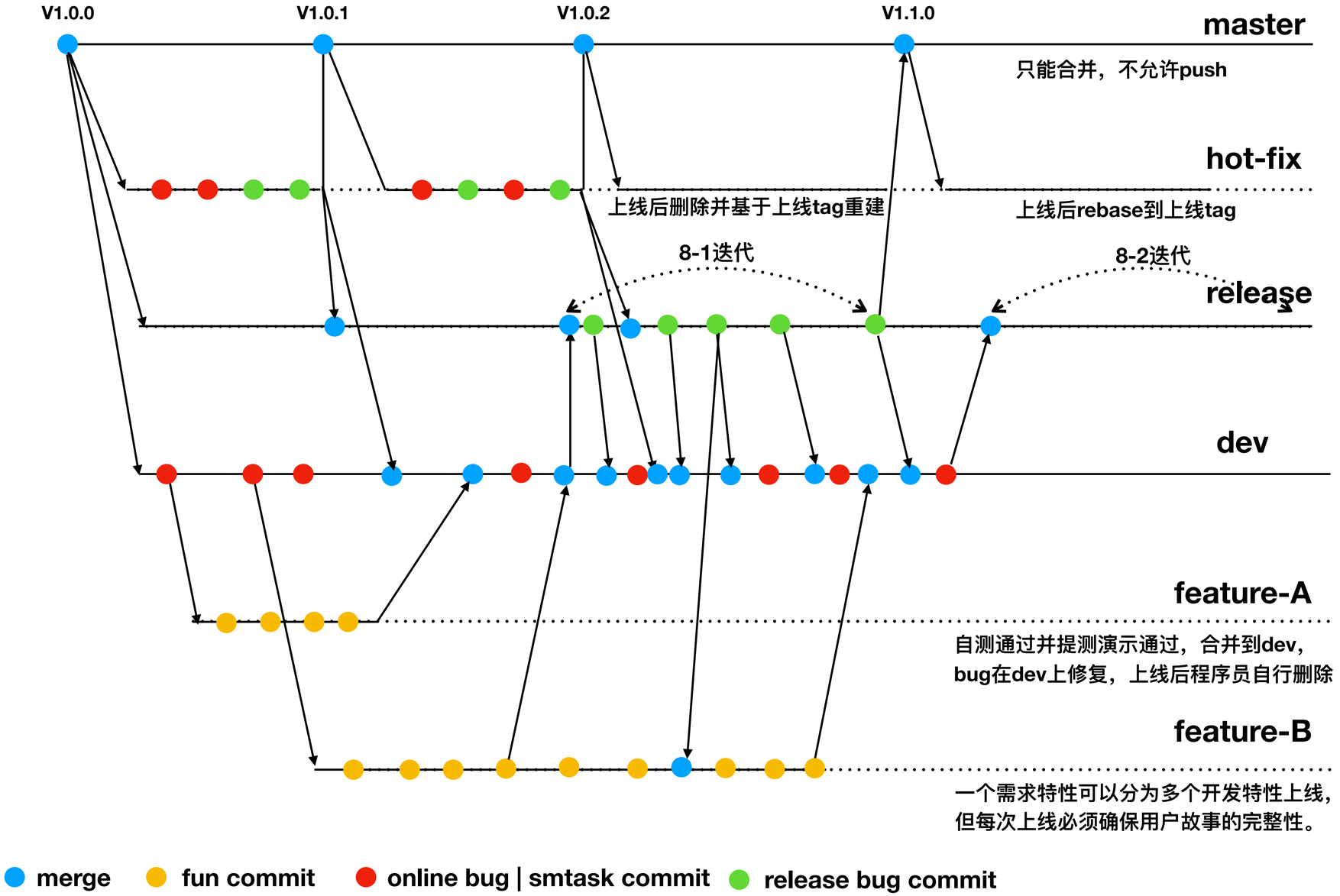
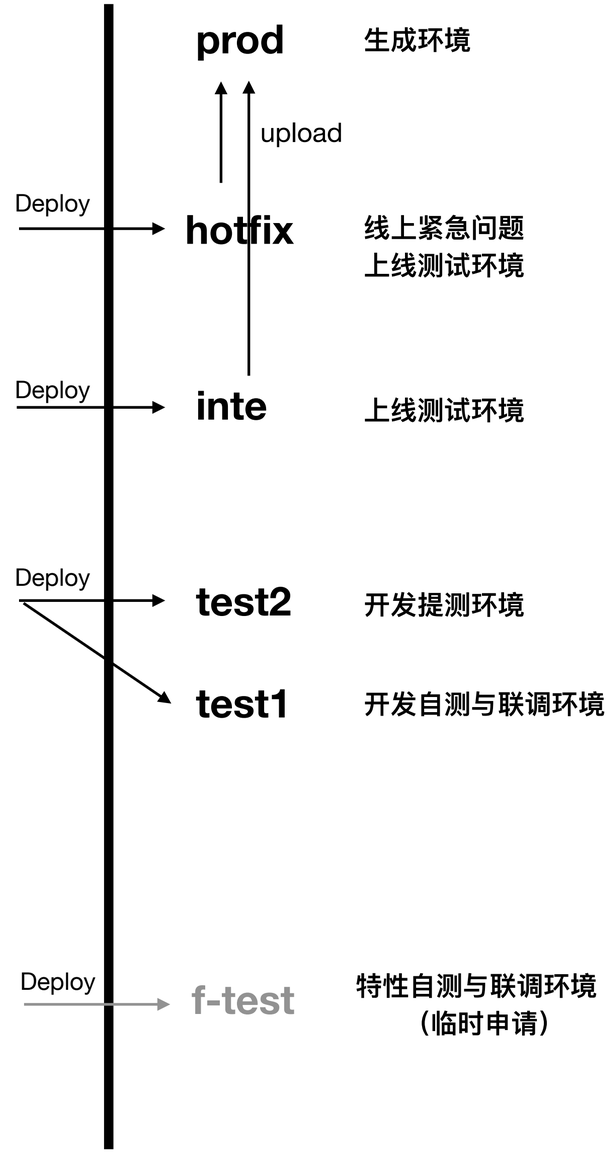
企企代码分支管理方案

总体说明

- 开发模式：敏捷迭代
- 迭代周期：2周
- 特性上线频率：每周四
- 线上紧急问题修复最小频率：天
- 版本管理工具：git
- 开发阶段划分：开发->测试->上线测试->上线(灰度->正式)
- 环境：test1（开发联调自测）、test2（开发提测）、inte（上线测试）、hot-fix（线上紧急问题修复开发测试）、prod（生产）、特性临时联调环境（有需要时临时申请）
- 代码分支：
  - master：与prod代码保持一致的代码基线
  - dev：达到提测标准的最新代码（开发周期小于1天的任务可以在dev上直接提交，但push时必须自测通过）
  - release：上线测试及上线过程的bug修复
  - feature-xx：迭代业务特性，业务特性上线后即可删除，一个特性可以在多个迭代中上线（开发细分上线迭代）

分支关系及生命周期图


V1.0.0
V1.0.1
V1.0.2
V1.1.0
master
只能合并，不允许push
hot-fix
上线后删除并基于上线tag重建
上线后rebase到上线tag
8-1迭代
8-2迭代
release
dev
feature-A
自测通过并提测演示通过，合并到dev，
bug在dev上修复，上线后程序员自行删除
feature-B
一个需求特性可以分为多个开发特性上线，
但每次上线必须确保用户故事的完整性。
merge
fun commit
online bug | smtask commit
release bug commit


prod
生成环境
upload
Deploy
hotfix
线上紧急问题
上线测试环境
Deploy
inte
上线测试环境
Deploy
test2
开发提测环境
test1
开发自测与联调环境
Deploy
f-test
特性自测与联调环境
（临时申请）

日常开发过程与行为规范

- 接收开发任务：
  - 业务特性：业务特性开发负责人从dev拉出特性分支，特性开发组成员都基于该分支开发，分支命名规范：feature-{特性代号[8个字符以内]}
  - 开发任务时间小于4小时的开发任务（需求或者线上非紧急bug），基于本地dev开发（push时必须自测通过）。
- 提测：
  - 业务特性：自测且演示通过，合并对应的feature分支到dev并push到远端（在邻近迭代上线测试期需要注意与开发负责人和测试负责人确认）。
  - 开发任务时间小于4小时的开发任务（需求或者线上非紧急bug），自测通过则push dev到远端 。
- 日常bug修复：
  - 在dev上修复，开发自己决定是否合并到feature分支。
- 开始迭代上线测试：
  - 每周五中午由前后端开发负责人与测试负责人确认dev分支是否达到上线测试标准（功能测试都必须通过），没有达到的督促达成（为了迭代上线顺利，每周四开始禁止合并开发周期超过3天的业务特性到dev）。
  - 每周五下班前，再次确认后状态后，前后端开发负责人合并dev到release，构造并部署inte环境（包括全新租户和基于线上版本的升级租户），确保能正常开展测试工作。
- 上线测试及bug修复：
  - 在release上修复，并合并到dev，开发自己决定是否合并到feature分支。
- 上线：
  - 测试达到上线标准，release合并到master，并打版本tag。
  - hot-fix分支rebase到新的上线tag。

hot-fix开发过程与行为规范

- 接收开发任务：
  - 线上紧急bug修复任务，pull hot-fix到本地，在本地hot-fix开发（push时必须自测通过）。
- 提测：
  - 自测通过则push 到hot-fix远端，并通知测试 部署验证。
- hot-fix测试及bug修复：
  - 在hot-fix上修复，自测通过则push到hot-fix远端，并通知测试 部署验证。
- 上线：
  - 测试达到上线标准，hot-fix合并到master，并打版本tag。
  - hot-fix合并到release分支和dev分支。
  - 删除hot-fix分支，并基于最新的上线tag重建hot-fix分支。
  - 如果某个feature分支也想看见这个hot-fix的代码，程序员自己决定rebase feature分支到dev分支最新点或者指定的提交点，也可以merge dev到feature分支。

日常构造部署及上线行为规范

- 开发自测及前后端联调环境Test：
  - 代码分支：dev　　环境：test1
  - 开发自己决定并在群里通知，执行dev构造和部署jenkins任务。
- 功能测试：
  - 代码分支：dev 环境：test2
  - 测试主管决定并在群里通知，执行test构造和部署jenkins任务
- 上线测试：
  - 代码分支：release　环境：inte
  - 测试主管决定并在群里通知，执行inte构造和部署jenkins任务
- 上线：
  - 测试主管和开发主管决定并在群里通知，执行release-jenkins任务（代码合并到master、打tag）。
  - 开发主管rebase hot-fix分支到上线tag。

hot-fix构造部署及上线行为规范

- hot-fix测试：
  - 代码分支：hot-fix 环境：hot-fix
  - 测试主管决定并在群里通知，执行hot-fix构造和部署jenkins任务。
- Hot-fix上线
  - 测试主管和开发主管决定并在群里通知，执行hot-fix-release-jenkins任务（代码合并到master，打tag）。
  - 开发主管合并hot-fix到release和dev。
  - 删除hot-fix分支，并基于最新的上线tag重建hot-fix分支。

git提交规范

- 提交时能知道需求编号的必须在提交注释中声明需求编号。
- 修复bug时必须声明bug号，如果上文知道需求编号，建议同时声明需求编号。
- 格式：git commit -m “[<pr-no[,pr-no…]>][-][<bug-no>]简短清晰的提交内容说明”
- 例子：
  - git commit -m “<pr-201>提测数据权限操作员权限”
  - git merge —no-ff -m “<pr-301>提测自定义档案作为辅助核算特性” feature-pr-301
  - git commit -m “<pr-101>-<bug-1401>修复自定义档案数据权限控制因素显示名称不正确”
  - git commit -m “<bug-2001>修复科目删除时没有控制已经填制凭证”

产品版本及git tag规范

- 版本格式：采用三段式版本号，d.d.d
  - 第一段：产品重大特性升级
  - 第二段：产品日常特性升级
  - 第三段：hot-fix升级

  比如：产品首版prod上线版本号为：1.0.0，

  1.0.0上线后hot-fix1版本号为：1.0.1，hot-fix2版本号为：1.0.2

  1.0.0上线后迭代一上线版本号：1.1.0

  1.1.0上线后hot-fix1上线版本号：1.1.1，hot-fix2版本号为：1.1.2
- 上线后合并到master后，git tag命名规范：版本号.yyyymmdd，（yyyymmdd：上线日期）

  比如：1.1.0上线后git tag：1.1.0.20190830
- Git tag 规范：
  - git tag -a <tag name> -m <message>

  比如：git tag -a 1.1.0.20190830 -m “迭代1 自定义辅助核算 等特性 ”

git常用命令参考

- git branch —list -a 查看所有分支信息
- git checkout -b <branch> 创建并切换到新分支
- git checkout -b <branch> origin/<branch> 基于远程分支创建一个本地分支
- git merge —no-ff -m”merge message” <branch> 关闭快速向前合并（建议采用,用图形工具的注意改选项参数）
- 有冲突是提示手动解决冲突，然后用git merge —continue，放弃git merge —abort。
- git merge —squash 把分支的所有提交合并成一个提交点到合并分支。（需要多次合并的分支不建议使用，因为解决完冲突后，以后再次合并时可能还需再次解决冲突）
- git branch -d <branch> 删除已经合并完成的分支
- git log 查看提交记录. git log —oneline 单行显示提交记录 git log —stat 显示提交文件变更摘要信息
- git log <file>查看某个文件的提交记录 git log —author=“xx” 查看某人的提交记录