Git 开发流程规范

谢乔

主题

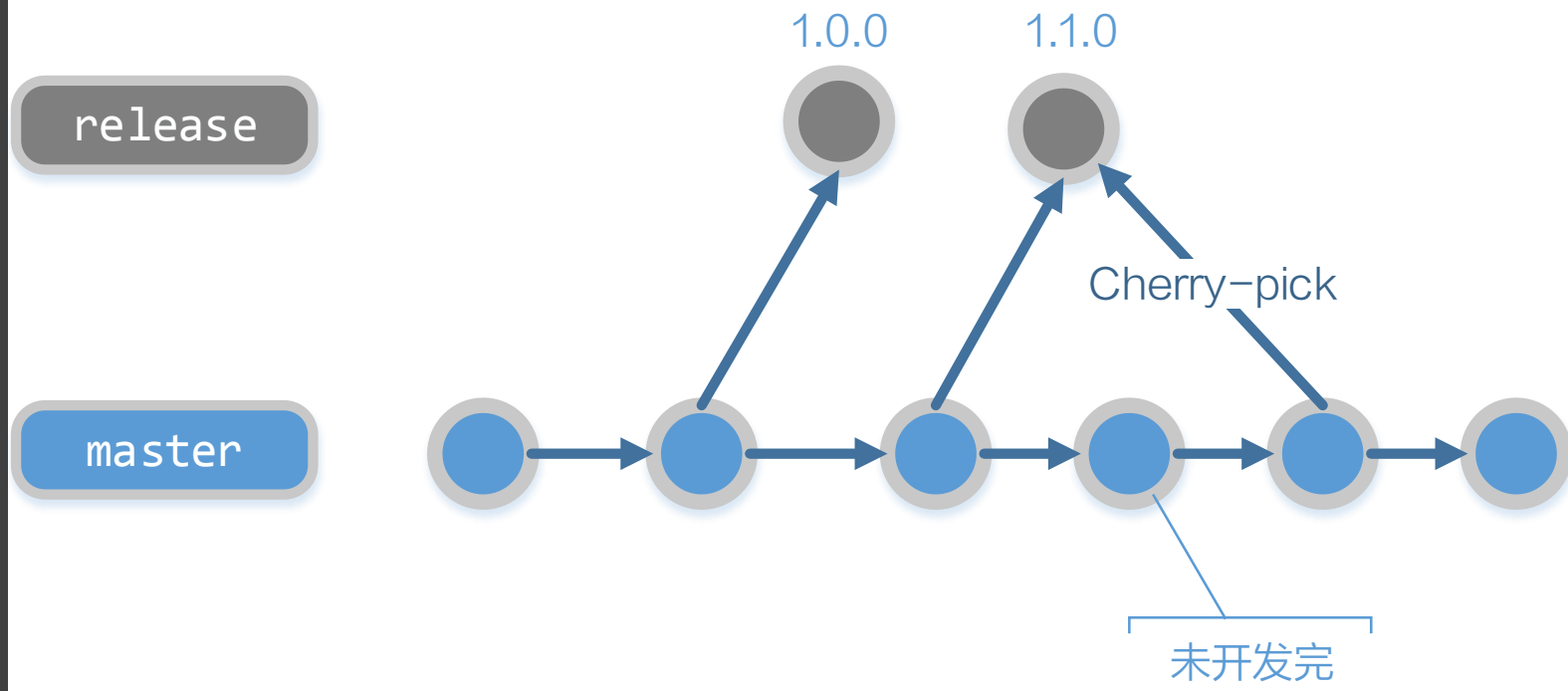
- 统一规范 git 分支开发的流程
- 不是 git 的使用说明

为什么要使用分支开发

这是一句废话吗?

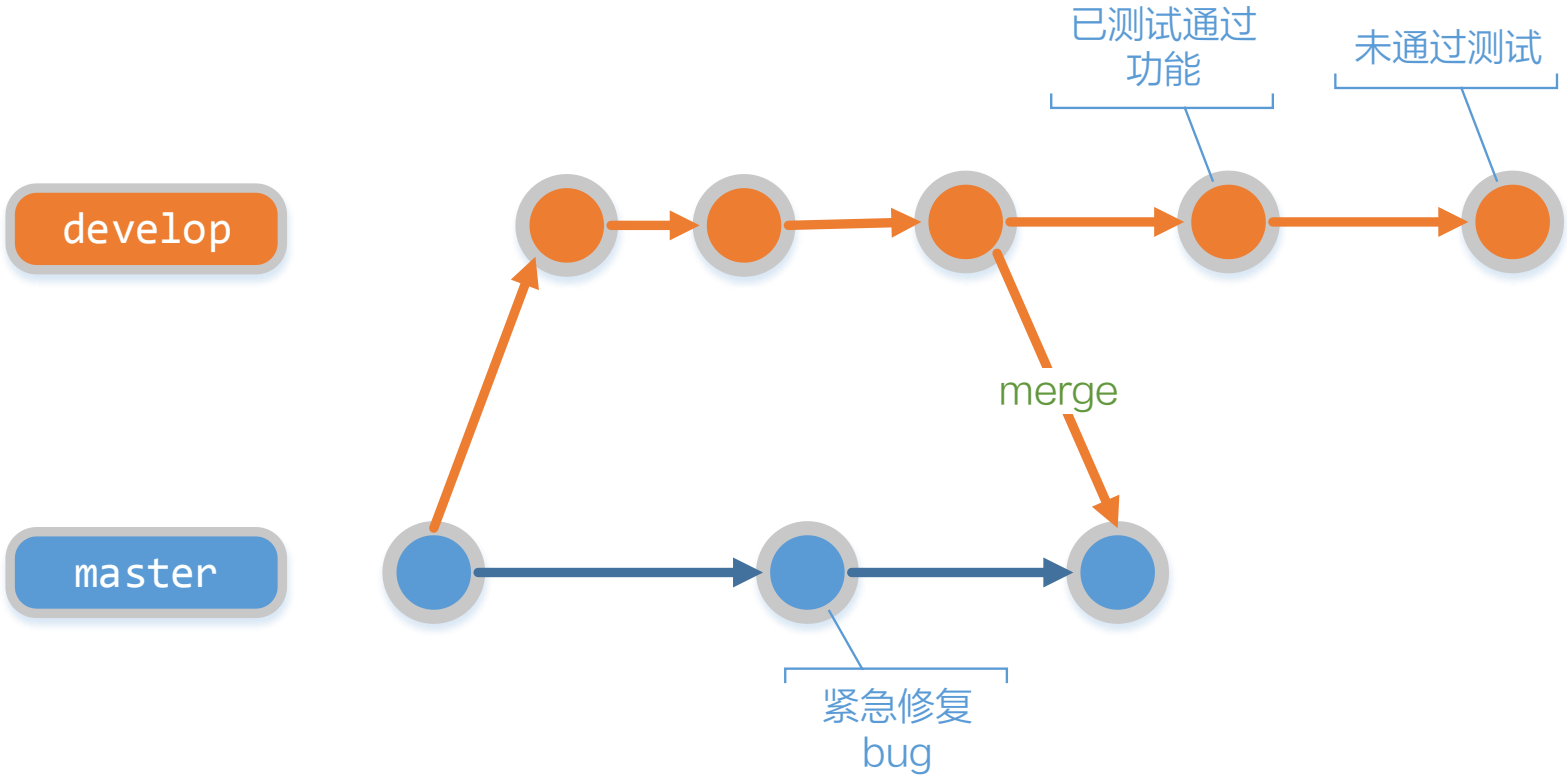
主干开发（TBD）

- 特点
  - 简单高效
  - 及时提交减少冲突
  - 发布分支是 master 的某个时间点快照
  - 适合小型项目或超级牛逼团队
- 问题
  - 未完成时，阻塞发布
  - 测试不友好
  - 紧急 bug 修复
  - 配合 cherry-pick，增加了出错的概率



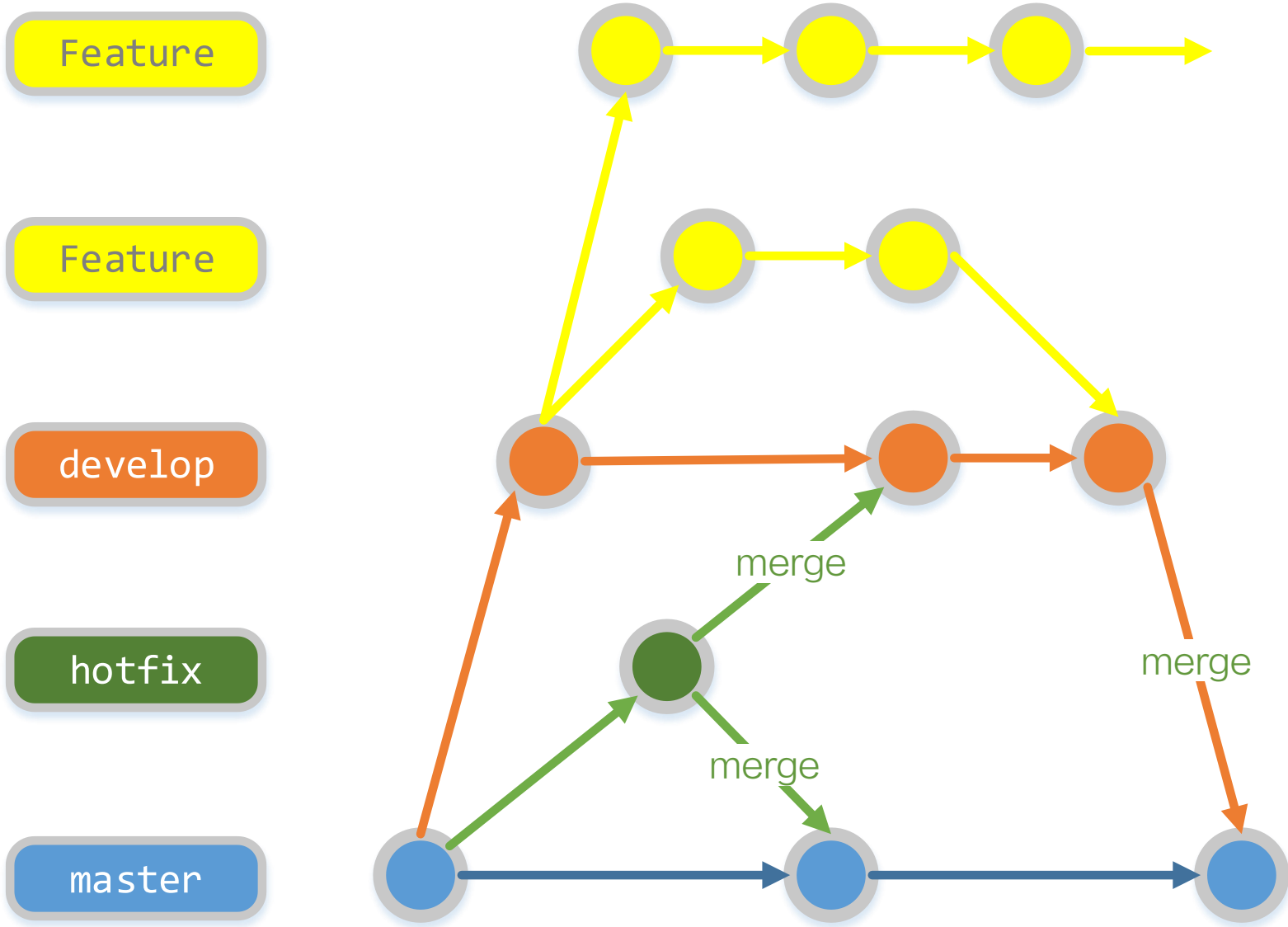
进化出一个 develop 分支

- 特点
  - master 保持与线上对应，可随时进行紧急 bug 修复
- 问题
  - 在 develop 分支上依然存在未完成代码，导致发布阻塞
  - 修复的 bug 没有及时同步到 develop
  - 测试不友好



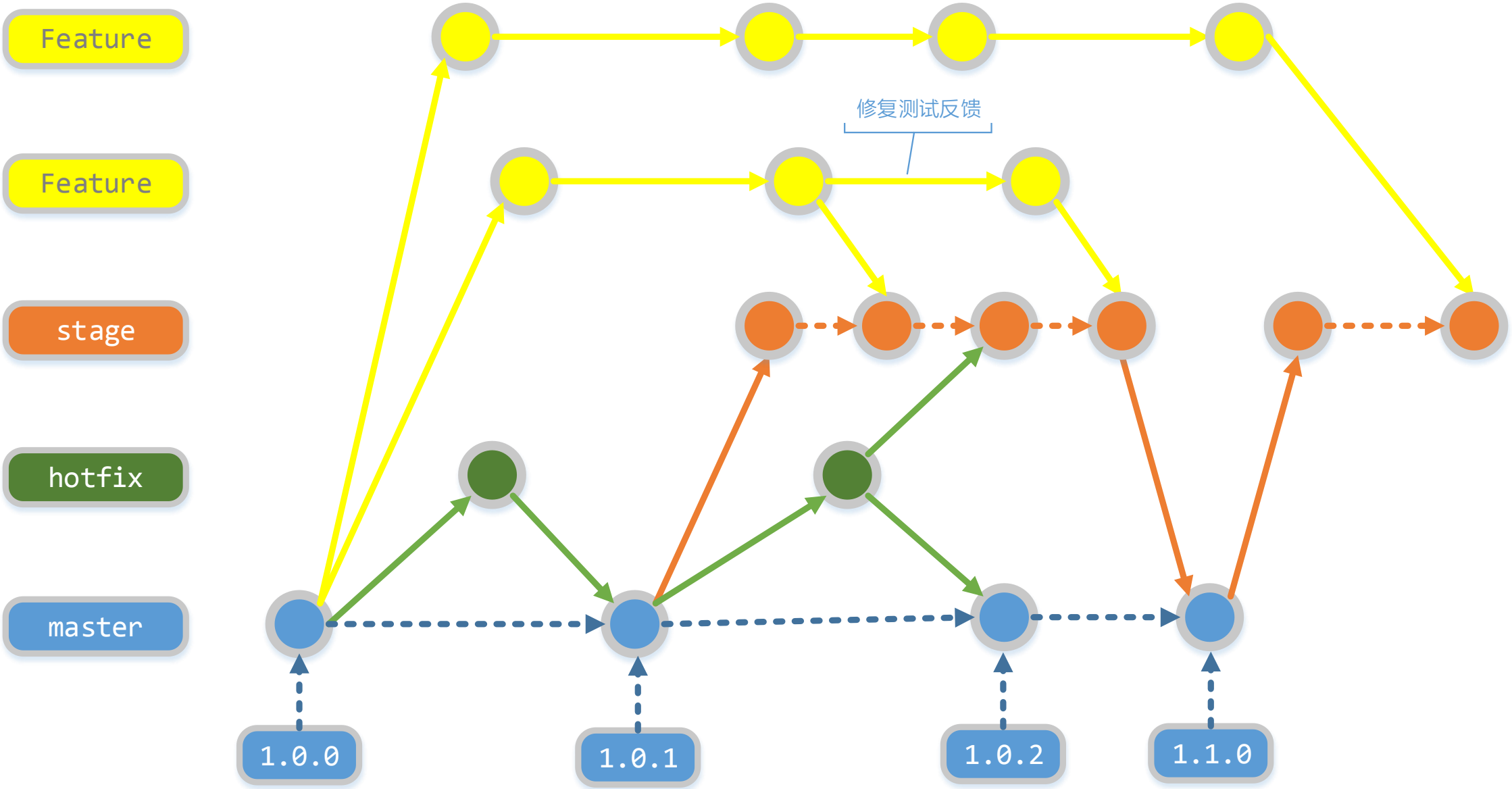
增加 hotfix 与 feature 分支

- 特点
  - 紧急修复的 bug 及时同步到所有分支
  - 发布不会被未完成的开发所阻塞
- 问题
  - 分支太多，没有规范将造成混乱
  - 冲突将变多，需要细心处理
  - 测试不友好



分支发开规范

KISS


Feature
Feature
stage
hotfix
master
修复测试反馈
1.0.0
1.0.1
1.0.2
1.1.0

分支类型

- master

有且仅有一个，对应线上版本，不能直接 commit。

- ~~develop (简化去掉)~~
- feature | bugfix

用于新功能或修复 bug，粒度按照一个原子功能创建一个分支。开发自测完毕后，merge 到 stage 分支进入 QA 测试阶段或预发布。

- hotfix

修复线上紧急 bug

- stage

为集成测试，避免边测试边开发带来的不便。一般情况只有一个，特殊场景允许有多个。

- tag

方便定位到指定版本，避免使用 rebase，防止 checkout 出中间状态的版本。

主流程规范

- master 只能用于创建分支和接收合并，严格对应线上代码，不能直接 commit。master 只能通过向前合并分支的方式对其修改，接收 stage 和 hotfix 的 merge request。
- feature 由 master 创建分支，分支保持原子性的功能。不能直接合并到 master。开发自测完毕后，先从 master 创建 stage 分支，再将 feature 合并到 stage，最后由 stage 合并到 master。
- stage 分支是为 QA 团队测试使用的，需要设定一个版本号，如 stage/1.3.0。在 stage 上测试出的 bug，在对应的 feature 分支上进行修复，最后在合并到 stage 进行复测。
- 使用 gitlab 的权限管理与 merge request。项目 leader 拥有 master 权限；开发人员为 developer 权限，只能通过 merge request 合并代码。leader 在 merge request 中进行 code review。
- 应对线上代码的紧急 bug 修复，需要从 master 创建 hotfix 分支。hotfix 上线后需要提交 merge request 合并到 master，同时要 merge 到所有 stage 分支。
- 最终通过 stage 上线，确认无误后提交 merge request 合并回 master和所有stage分支。同时从 master 创建 tag，名称使用版本号，如1.4.0。创建 tag 时，务必使用 md 填写 Release notes。
- 及时清理掉已完成的 feature、hotfix、stage 分支。

禁止行为

- 禁止 master reset，master 只能向前发展。
- 禁止 stage 直接修改代码。
- 禁止 stage 向 feature 合并。
- 禁止 feature 直接向 master 合并。

大家想想这些限制是为了什么？

特殊情况

1. 如果多个 feature 合并到 stage 进行测试，部分 feature 没有通过测试，可以新建 stage 集成已测试通过的 feature，保障最大努力上线已完成的功能。
2. feature 开发周期过长时，可以从 master 向该 feature 合并。
3. 团队成熟度很高，人人都可拥有 master 权限。

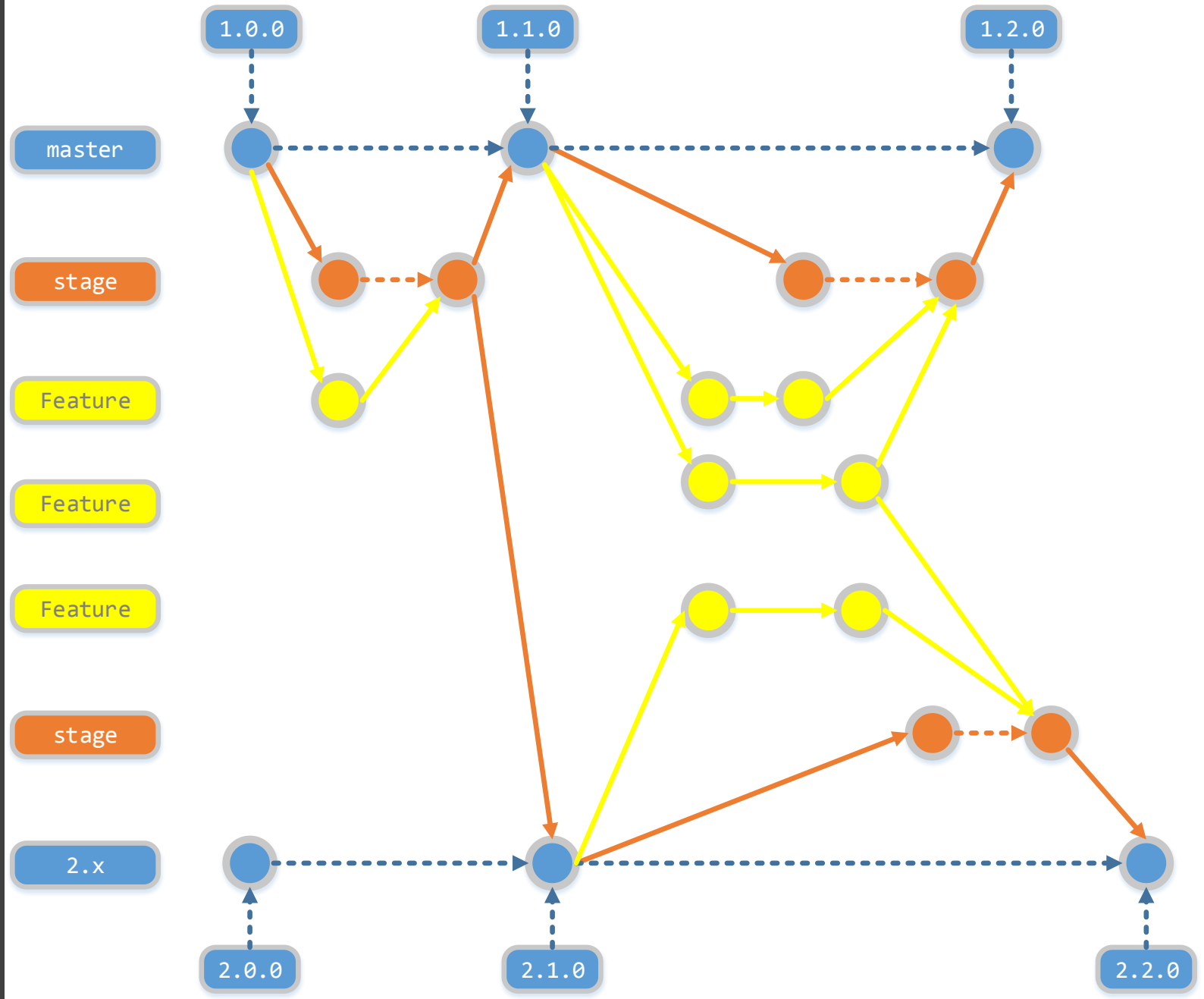
以上所有出现时，都不得违背**[禁止行为]!!!**

尽量避免

- cherry-pick
- reset
- patch
- rebase
- 多版本并行开发

三界之外的多版本并行开发

- 只能破戒了。具体问题具体分析。
- 创建一个并行的与master同级别的版本分支，遵循master的规范，使用版本号为分支名，例如“2.x”。
- 按实际分析，哪些feature或hotfix分支需要双向合并。



总结

- 规范不是目的，目的是真正理解分支开发解决什么问题。
- 规范不是万能的，有时候会带来不便，能有效降低出现低级错误的概率。
- 规范不是为了约束人，让规范成为统一的习惯，最终提升效率。
- 如果能简化，那就继续简化这个规范。

changelog

```
# 更新日志

## <version> - <date>

### <type>
* <desc>
* <desc>

### <type>
* <desc>
* <desc>
```

- 推荐在项目中添加Changelog.md
- type的值有4种，feature、bug fix 、change、deprecated。