

1、Git常用命令

[Git使用介绍](#) [Git使用规范](#)

1.1、Git常用命令对比与使用标准

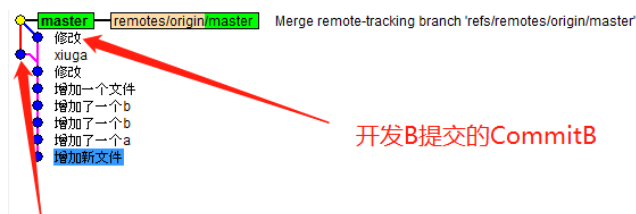
① 将远程代码拉到本地（非合并）

```
1 | git fetch;
```

② 合并代码（需要先通过fetch拉到本地，不然本地不知道远程仓库的情况）

```
1 | git rebase; // 指将当前分支远程的合并到工作目录
2 | git rebase fb_20200526_dev; // 将fb_20200526_dev分支合并到当前分支
3 | git merge; // 指将当前分支远程的合并到工作目录
4 | git merge fb_20200526_dev; // 将fb_20200526_dev分支合并到当前分支
5 | git pull; // 相当于同时执行git fetch; git merge; 一般情况下禁止使用！
```

总结：小组开发时，同分支更新代码采用**rebase**，不同分支代码合并采用**merge**

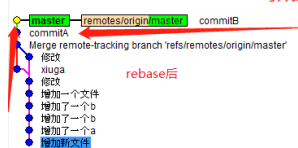


开发B提交的CommitB

开发A提交的一个CommitA

使用merge的情况

小组开发的时候，在同一个开发分支上，开发A提交了一个commitA到仓库之后开发B提交了一个commitB准备推送到远程，此时Git会提示让你先更新。这个时候通过git merge来更新，就会基于commitA与commitB的合并结果生成一个新的commitC，在关系图上就可以看出分叉开后，又合并在了一起了！



开发A提交的CommitA

rebase后

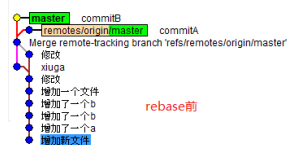
开发B提交的commitB

使用rebase的情况

shuang_gu <shuang_gu@kingdee.com>	2020-05-26 16:08:11
shuang_gu <shuang_gu@kingdee.com>	2020-05-26 16:07:50
shuang_gu <shuang_gu@kingdee.com>	2020-05-26 16:01:44
shuang_gu <shuang_gu@kingdee.com>	2020-05-26 16:01:07
shuang_gu <shuang_gu@kingdee.com>	2020-05-26 16:01:31
shuang_gu <shuang_gu@kingdee.com>	2020-05-26 16:00:02
shuang_gu <shuang_gu@kingdee.com>	2020-05-26 15:39:33
shuang_gu <shuang_gu@kingdee.com>	2020-05-26 15:35:52
shuang_gu <shuang_gu@kingdee.com>	2020-05-26 15:08:41
shuang_gu <shuang_gu@kingdee.com>	2020-05-26 15:04:35
古贤 <shuang_gu@kingdee.com>	2020-05-26 16:28:39

采用rebase的时间也是很有规律的

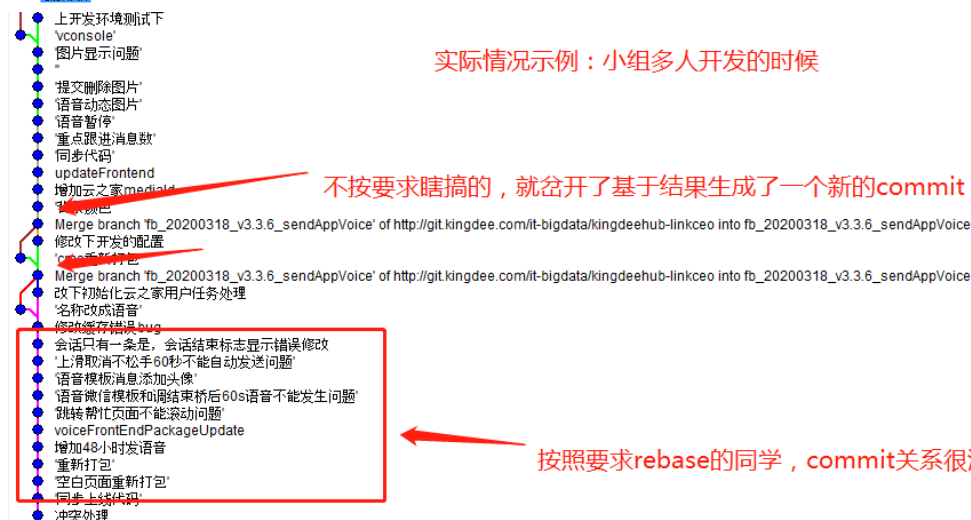
当使用rebase的时候，Git会将远程的新增commit作为你的提交基地，如图，你是开发B你提交了一个commitB此时你提交的时候告诉你远程有更新需要先更新，此时使用rebase更新会将远程的commitA作为你的基点更新，更新前基点为那个叫Merge XXX的commit。此时生成的关系图为一严格的直线！



rebase前

实际情况示例：小组多人开发的时候

不按要求瞎搞的，就岔开了基于结果生成了一个新的commit



按照要求rebase的同学，commit关系很清晰

一种更糟糕的情况

当然导致了这种情况也并不仅仅是merge的滥用



③ 将文件交于Git管理

- 1 | `git add xxx.txt; //` 将xxx.txt文件交于git做版本管理
- 2 | `git add .; //` 将当前目录下的所有文件交于git管理

④ 提交新的commit

- 1 | `git commit -m "提交描述"; //` 其他详细参数请查阅`git --help`

⑤ 将commit推到仓库

- 1 | `git push;`

⑥ 打tag

- 1 | `git tag tag_20200526_xxx; //` 在基于当前commit打一个tag
- 2 | `git push origin tag_20200526_xxx; //` 将当前tag推送到远程仓库

⑦ 切分支

- 1 | `git checkout develop; //` 切换到新develop分支
- 2 | `git checkout -b hotfix_20200526_fixBug; //` 基于当前commit切出一个名为hotfix_20200526_fixBug的分支
- 3 | `git push origin --set-upstream hotfix_20200526_fixBug; //` 将hotfix_20200526_fixBug分支推送到仓库并与本地的hotfix_20200526_fixBug分支关联

⑧ 暂存

- 1 | `git stash; //` 将本地修改暂存到栈
- 2 | `git stash pop; //` 将栈中暂存的内容弹出
- 3 | `git stash list; //` 查看栈中有哪些暂存

⑨ 分支管理

- 1 | `git branch -a; //` 查看仓库所有的分支

1.2、最佳实践

① 更新远程代码到本地 场景1：本地没有未push的commit，本地无代码修改

The screenshot shows a terminal window and a Git GUI. The terminal window displays the following commands and output:

```
HSZC1712-0392+Administrator@HSZC1712-0392 MINGW64 /h/git测试/project-b - 副本 (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.
nothing to commit, working tree clean
```

A red arrow points to the `git status` command, and a red text box says "查看当前状态，提示本地是干净的" (Check current status,提示本地是干净的).

```
HSZC1712-0392+Administrator@HSZC1712-0392 MINGW64 /h/git测试/project-b - 副本 (master)
$ git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From http://git.kingdee.com/shuang_gu/project-b
d2d1699..a9a1326 master -> origin/master

HSZC1712-0392+Administrator@HSZC1712-0392 MINGW64 /h/git测试/project-b - 副本 (master)
$ git rebase
First, rewinding head to replay your work on top of it...
Fast-forwarded master to refs/remotes/origin/master.

HSZC1712-0392+Administrator@HSZC1712-0392 MINGW64 /h/git测试/project-b - 副本 (master)
$
```

The Git GUI shows the commit history. A red arrow points to the `master` branch, and a red text box says "查看Git commit关系图可知当前head与远程的master指向了同一个commit 证明当前代码已经与远程对应分支相同" (Check the Git commit relationship diagram, it can be seen that the current head and the remote master point to the same commit, proving that the current code is the same as the remote corresponding branch).

The commit history shows a single commit: `a9a1326f78137ea96e9e131694177c86fb3ea615` (增加了一个a). The commit message is "增加了一个a".

- 1 git fetch;
- 2 git rebase;

场景2：本地没有未push的commit，本地有代码修改

```
HSZC1712-0392+Administrator@HSZC1712-0392 MINGW64 /h/git测试/project-b - 副本 (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   b.txt

MINGW64:/h/git测试/project-b - 副本
$ git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From http://git.kingdee.com/shuang_gu/project-b
   a9a1326..6b62ab4  master    -> origin/master

$ git rebase
Cannot rebase: Your index contains uncommitted changes.
Please commit or stash them.

MINGW64:/h/git测试/project-b - 副本
$ git stash
Saved working directory and index state WIP on master: a9a1326 增加了一个a

$ git fetch
git
$ git rebase
First, rewinding head to replay your work on top of it...
Fast-forwarded master to refs/remotes/origin/master.

$ git stash pop
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   c.txt

Dropped refs/stash@{0} (70b35fb4175fe5f7ced36995d0532e74d571f0f8)

MINGW64:/h/git测试/project-b - 副本
$
```

查看状态提示本地有修改

执行git fetch提示有更新
执行git rebase提示本地有修改，请先暂存

正确的玩法应该是

先将本地修改暂存

将代码拉到本地并更新

将之前暂存的内容从栈中弹出



```
1 git stash;  
2 git fetch;  
3 git rebase;  
4 git stash pop;
```

场景3：本地有未push的commit，本地无代码修改

```
HSZC1712-0392+Administrator@HSZC1712-0392 MINGW64 /h/git测试/project-b - 副本 (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
(use "git push" to publish your local commits)

nothing to commit, working tree clean

HSZC1712-0392+Administrator@HSZC1712-0392 MINGW64 /h/git测试/project-b - 副本 (master)
$ git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From http://git.kingdee.com/shuang_gu/project-b
6b62ab4..06145c9 master -> origin/master

HSZC1712-0392+Administrator@HSZC1712-0392 MINGW64 /h/git测试/project-b - 副本 (master)
$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commits each, respectively.
(use "git pull" to merge the remote branch into yours)

nothing to commit, working tree clean
```

查看状态提示当前比远程有一个更新的commit，请推到远程

将远程代码拉下来

再次查看状态提示有分叉

The screenshot shows a terminal window with the following commands and output:

```
HSZC1712-0392+Administrator@HSZC1712-0392 MINGW64 /h/git测试/project-b - 副本 (master)
$ git status
git: 'git' is not a git command. See 'git --help'.

The most similar command is
  init

HSZC1712-0392+Administrator@HSZC1712-0392 MINGW64 /h/git测试/project-b - 副本 (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
(use "git push" to publish your local commits)

nothing to commit, working tree clean

HSZC1712-0392+Administrator@HSZC1712-0392 MINGW64 /h/git测试/project-b - 副本 (master)
$ git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From http://git.kingdee.com/shuang_gu/project-b
6b62ab4..06145c9 master -> origin/master

HSZC1712-0392+Administrator@HSZC1712-0392 MINGW64 /h/git测试/project-b - 副本 (master)
$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commits each, respectively.
(use "git pull" to merge the remote branch into yours)

nothing to commit, working tree clean

HSZC1712-0392+Administrator@HSZC1712-0392 MINGW64 /h/git测试/project-b - 副本 (master)
$ gitk --all
gitk --all
```

The gitk graphical interface shows a commit history with a divergent branch. The local master branch has a commit labeled "增加一个文件". The remote origin/master branch has a commit labeled "增加了一个b". The commit history shows a sequence of commits: "增加了一个b", "增加了一个a", and "增加新文件".

查看图可知确实有分叉，即自己本地有一个commit，远程也有一个commit。这两个commit基于同一个commit版本岔开

```
HSZC1712-0392+Administrator@HSZC1712-0392 MINGW64 /h/git测试/project-b - 副本 (master)
$ git rebase
First, rewinding head to replay your work on top of it...
Applying: 增加一个文件
```

此时直接rebase即可

The screenshot shows a terminal window with the following commands and output:

```
HSZC1712-0392+Administrator@HSZC1712-0392 MINGW64 /h/git测试/project-b - 副本 (master)
$ git rebase
First, rewinding head to replay your work on top of it...
Applying: 增加一个文件
```

The gitk graphical interface shows the commit history after the rebase. The local master branch now has a commit labeled "增加一个文件". The remote origin/master branch has a commit labeled "增加了一个b". The commit history shows a sequence of commits: "增加了一个b", "增加了一个a", and "增加新文件".

刚刚的分叉合并到一起了

```
1 git fetch;
2 git rebase;
```

场景4: 本地有未push的commit, 本地有代码修改(大家可以按照上面的套路, 自行尝试)

```
1 git stash;
2 git fetch;
3 git rebase;
4 git stash pop
```

② 正常情况下的全流程介绍

```
1 git tag; // 找到最新的tag
2 git stash; // 暂存本地修改(看实际情况可选)
3 git checkout tag_xxx; // 切到最新tag, 实际上就是一个commit
4 gitk --all; // 看一下图是否head已经指向远程的最新tag, 以及与远程的master关系。
5 git checkout -b hotfix_20200526_bugfix; // 基于tag切一个bug修复分支
6 git push origin --set-upstream hotfix_20200526_bugfix; // 将分支推到仓库并与本地关联
7 ... 改代码
8 git checkout test;
9 git fetch;git rebase; // 切到test分支并更新
10 gitk --all; // 核对更新结果
11 git merge hotfix_20200526_bugfix; // 将bug修复分支合并到test分支测试(不同分支合并采用merge)
12 gitk --all; // 核对合并结果
13 git push;
14 ... 测试完毕发布生产环境
15 git checkout master;
16 git fetch;git rebase; // 切到master分支并更新
17 gitk --all; // 查看master分支是否为最新, 是否打了tag
18 git merge hotfix_20200526_bugfix; // 站在master分支上合并修复分支
19 gitk --all; // 查看合并结果是否正确
20 git push;
21 ... 发布更新并验证通过
22 git tag tag_20200526_xxxx;
23 git push origin tag_20200526_xxxx; // 将tag推仓库
```

1.3、日常学习方式

① 除非你对Git很熟悉, 不然少用ide的Git相关按钮! 即便要用也看清楚相关选项, 别用默认的, 更新默认的就是使用merge! 注: 使用ide集成的Git更新代码的时候没注意观察ide的进度, 你会有惊喜的发现。其实他的做法就是

```
1 git fetch;
2 git stash;
3 git rebase;
4 git stash pop;
```

有兴趣的同学可以自行观察!

② 善于使用git bash, 通过指令操作Git!

- 好处1: 能够熟悉Git相关操作, 熟悉Git执行流程
- 好处2: 在linux环境中只能通过指令操作

- 好处3: 在win平台上git bash就是一个简单的linux环境, 能够用常用的工具, 例如curl、vim等, 同时支持linux shell脚本执行。

③ 如遇忘记指令等情况, 或者不知道参数如何写

- 推荐通过git -help

```
HSZC1712-0392+Administrator@HSZC1712-0392 MINGW64 /d/kingdeehub-linkceo (master)
$ git branch -help
usage: git branch [<options>] [-r | -a] [--merged | --no-merged]
or: git branch [<options>] [-l] [-f] <branch-name> [<start-point>]
or: git branch [<options>] [-r] (-d | -D) <branch-name>...
or: git branch [<options>] (-m | -M) [<old-branch>] <new-branch>
or: git branch [<options>] (-c | -C) [<old-branch>] <new-branch>
or: git branch [<options>] [-r | -a] [--points-at]
or: git branch [<options>] [-r | -a] [--format]

Generic options
  -v, --verbose          show hash and subject, give twice for upstream branch
  -q, --quiet            suppress informational messages
  -t, --track            set up tracking mode (see git-pull(1))
  -u, --set-upstream-to <upstream>
                        change the upstream info
  --unset-upstream      Unset the upstream info
  --color[=<when>]      use colored output
  -r, --remotes          act on remote-tracking branches
  --contains <commit>  print only branches that contain the commit
  --no-contains <commit>
                        print only branches that don't contain the commit
  --abbrev[=<n>]        use <n> digits to display SHA-1s

Specific git-branch actions:
  -a, --all              list both remote-tracking and local branches
  -d, --delete          delete fully merged branch
  -D                    delete branch (even if not merged)
  -m, --move            move/rename a branch and its reflog
```

- 其次再是百度, 注意百度来的指令请自行验证, 不要污染了仓库

④ 善用gitk --all指令

- 查看commit、分支关系
- 分析代码是否合并正确

注: 建议使用ide操作Git的场景

- 提交代码: 因为需要选择提交文件, 提交内容。git bash操作不便
- 解决冲突: git bash容易出错, 不直观

2、KEBC仓库拆分

2.1、当前KEBC仓库情况

仓库：所有模块代码全部放在一个Git仓库中管理，不同的业务采用不同的目录划分，他们之间没有任何关系！

it-bd > next > Repository


develop

next / kded / +

History

Find file

Web IDE

 Merge remote-tracking branch 'origin/fb_20200514_v1.0.0_partner_lifeCycle_move' into develop
shuang_gu authored 21 minutes ago

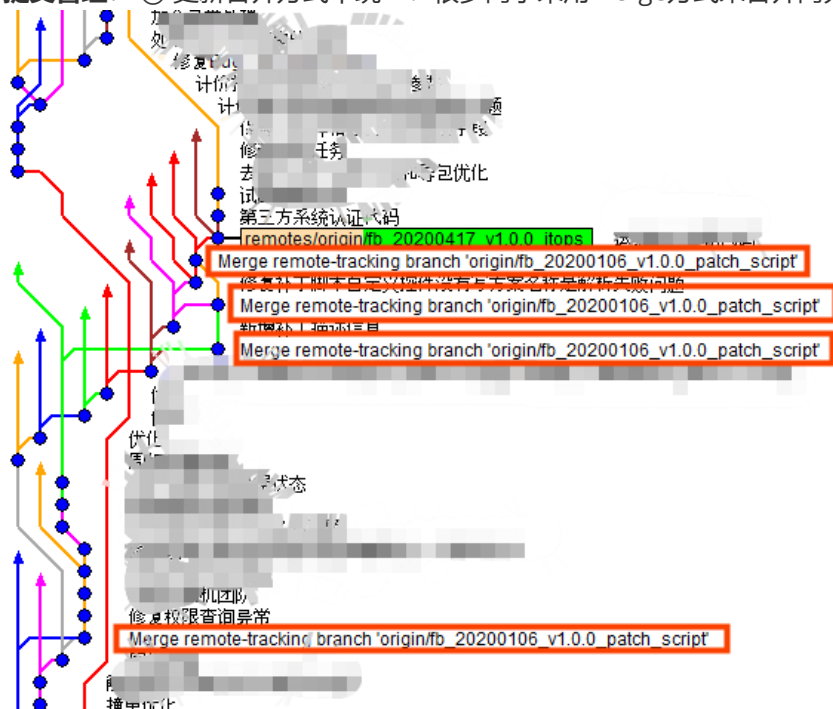
8810e8ce

Name	Last commit	Last update
..		
base	Merge branch 'fb_20200514_v1.0.0_partner_lifeCycle_m...	2 hours ago
cicd	新增获取发布环境获取审批人插件	1 week ago
data	伙伴档案全生命周期分支合并	2 weeks ago
es	Merge branch 'develop' of rdgit.kingdee.com:it-bd/next...	4 months ago
fi	Merge branch 'develop' of rdgit.kingdee.com:it-bd/next...	4 months ago
hr	修改医务室主页跳转方式	4 months ago
its	优化	1 month ago
kebc	Merge remote-tracking branch 'origin/fb_20200514_v1....	21 minutes ago
light/kded-light-health	API管理类：保留原始Map参数	2 months ago
temp/kded-temp-demo	新增模板工程	5 months ago

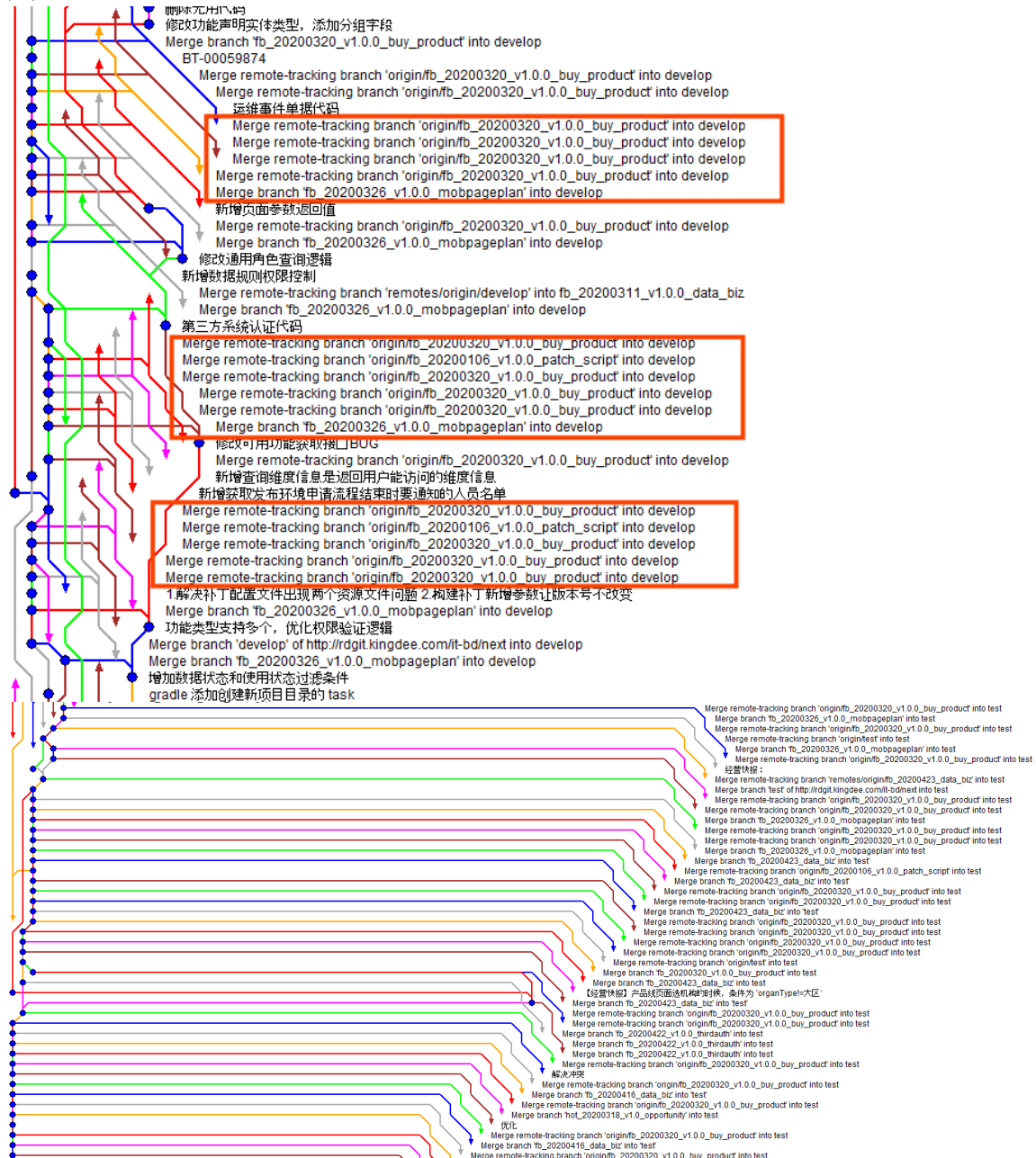
分支管理：各种业务分支都存在，有线索、伙伴、健康打卡、产品等等，他们之间没有任何的交集，完全独立，甚至能独立部署运行！

```
MING64: /d/IdeaProjects/next
20200209_gushuang
develop
fb_20200108_v1.0.0_video_center_change
fb_20200202_health
fb_20200207_health_v1.0
fb_20200208_health_v1.0
fb_20200310_mobile_file_upload
fb_20200320_v1.0.0_buy_product
fb_20200326_v1.0.0_mobpageplan
fb_20200422_v1.0.0_partner_lifeCycle
fb_20200512_v1.0.0_partner_lifeCycle
* fb_20200514_v1.0.0_partner_lifeCycle_move
gushuang
hotfix-20200227_fileservice
hotfix-20200302_file
hotfix-20200305_file
hotfix-20200310_filePluginFix
hotfix-20200327_file
hotfix-20200511_fileServiceGradleBugFix
lead_fb20191227_v1.0.0
master
temp
remotes/origin/HEAD -> origin/develop
remotes/origin/develop
remotes/origin/fb_20191207_v.1.0.0_addProductManagerFunction
remotes/origin/fb_20191211_v1.0.0_productManager
remotes/origin/fb_20191211_v1.0.1_product
remotes/origin/fb_20191213_v1.0.0_lead
remotes/origin/fb_20191224_v1.0.0_develop_temp
remotes/origin/fb_20191225_v1.0.0_customer
remotes/origin/fb_20191225_v1.0.0_file_service
remotes/origin/fb_20191227_v1.0.0_customerArchives
remotes/origin/fb_20191227_v1.0.1_product
remotes/origin/fb_20200101_customer
remotes/origin/fb_20200103_v1.0.0_fileUpload
remotes/origin/fb_20200106_v1.0.0_patch_script
remotes/origin/fb_20200108_v1.0.0_video_center_change
remotes/origin/fb_20200202_health
remotes/origin/fb_20200203_customer
remotes/origin/fb_20200205_health
remotes/origin/fb_20200207_healthReport
remotes/origin/fb_20200207_health_v1.0_warn
remotes/origin/fb_20200207_health_v1.0
remotes/origin/fb_20200208_health_v1.0
remotes/origin/fb_20200212_v1.0.0_partner_customer
remotes/origin/fb_20200215_health_v2.0
remotes/origin/fb_20200220_v1.0.0_cicd
remotes/origin/fb_20200227_Fileupdateapi
remotes/origin/fb_20200301_apimanage
```

提交管理：① 更新合并方式不统一：很多同学采用merge方式来合并同分支更新！



② 多业务开发团队同时并频繁操作同一个分支，导致commit关系图错综复杂！图例：合并到develop分支



版本管理：暂时还无tag，如果要对发布的稳定版本打tag，只能对整个大工程打tag，tag无法细分，例如prm发版想对prm单独打一个tag，无法实现。各个开发小组上线的版本tag会全部混在一起。

2.2、KEBC仓库拆分后目标

仓库：相关联的内容放在一个仓库中，例如prm仓库，里面就维护prm的内容，也很少会出现多个团队同时对同一仓库内容进行开发。

分支管理：例如prm开发小组基于prm仓库进行开发，也有自己的develop、test、master分支。容易：基于小组来管理Git。困难：基于大部门，甚至是多部门的开发同事。

提交管理：① 统一小组规则，具体参照部门的Git使用规则。由小组长监督！② 合并develop或者test分支由小组长统一合并！这样可以大幅缓解Git管理上的困难，图请不清晰好不好看直接取决于小组长有没有监督好组员。

版本管理：每个仓库（模块）单独进行版本管理，单独打tag。如果发布线上出现问题，也只知道还原这个模块即可，其他模块无需还原！

2.3、仓库拆分规则

模块名称	模块缩写	备注
客户关系管理	crm	对应现有kebc/crm
客户订单管理	cso	对应现有kebc/cso
内部订单管理	sal	暂无
项目交付管理	pms	暂无
伙伴关系管理	prm	对应现有kebc/prm
采购管理	pur	暂无
人力资源	hr	对应现有hr
财务类	fi	对应现有fi
it服务类应用	its	对应现有its
数据服务	data	暂无
基础资料（产品、客户、定价、物料、伙伴等等）	basicdata	暂无
基础服务（基础服务、登录、附件等等）	common	对应现有base

2.4、拆分后变化

如下以prm示例 clone工程：

```

1  git clone git@rdgit.kingdee.com:it-bd/prm.git; // 将prm工程拉下来
2  cd prm; // 进入到prm工程
3  git submodule init; // 初始化工程子模块，例如prm依赖了base，则会将base作为子模块拉下来
4  git submodule update; // 将依赖的子模块更新到最新（这里的最新看远程仓库指向子模块的哪个commit）

```

本地目录结构示例：

名称	修改日期	类型	大小
otherDevelop	2020-05-21 9:55	文件夹	
project-a	2020-05-21 11:13	文件夹	
project-b	2020-05-26 16:07	文件夹	
project-b - 副本	2020-05-26 16:08	文件夹	
project-base	2020-05-21 11:20	文件夹	
project-full	2020-05-20 17:24	文件夹	
project-prm	2020-05-27 14:56	文件夹	

clone下来后会会有一个prm的工程目录

名称	修改日期	类型	大小
.git	2020-05-27 14:55	文件夹	
project-base	2020-05-27 14:55	文件夹	
.gitmodules	2020-05-21 11:13	文本文档	1 KB
a.txt	2020-05-21 11:13	文本文档	0 KB

内部存在依赖的子模块的目录，.gitmodules文件记录了子模块依赖规则，.git目录里记录了对于子模块的指针（指向哪个commit）

2.5、拆分后的注意事项以及问题

注意事项：① 因为采用了子模块的概念，而git是基于commit来管理的，说白了就是主工程所依赖的子模块可以是任意的一个commit。假如说base模块的最新tag是tag2，但是prm当前还是指向base的上一个tag1。则prm开发负责人需要在prm下次发版的时候将依赖的子模块指向tag2，保证及时发现问题并暴露，使用最新的稳定本子模块。这样势必会导致操作的复杂与繁琐，但是为了管理的准确与清晰，这点代价是值得的！

② 假如A模块依赖B模块。B模块今天晚上7点发版，然后发版验证成功并打了tag，8点钟生产环境爆出问题。而恰巧A模块也在今天发版，发版的时候用了最新的tag，导致A模块发布失败！对于这种情况需要B维护团队通知到所有依赖方，在最新版本B模块基础上做评估与测试，以保证不会影响依赖方。而不是闷声发大财，啥也不管，发了再说！因为假如B模块更新了，但是A模块还是指向了B模块的上一个tag，本地开发环境是无法发现任何异常的。当发版的时候强制使用B模块的最新tag构建，就会暴露出问题，但是那个时候暴露出问题就晚了。

注：多仓库依赖采用git submodule实现 拆分测试完毕后会提供一份详细的操作文档，详询伟大的架构师！[官网文档连接](#)