配置管理规范

（基于 Git 代码管理）

密级： ☐ **保密** ☒ **秘密** ☐ **机密** ☐ **绝密**

文档编号：

拟制者	韩亚平	2013-3-22
审核	刘剑、谢东	
批准	谢东	

北京用友集团 UAP 中心

2013 年 3 月

版本修订记录

编号	修订日期	版本	修订人	修 订 内 容
1	2013-03	V0.1	刘剑	提供 PPT 文档，为编制规范基础文件
2	2013-03-25	V0.2	韩亚平	根据刘剑提供 PPT 进行梳理形成规范初稿
3	2013-03-28	V0.3	韩亚平	根据谢东总意见修订
4	2013-05-07	V0.4	刘剑	
5	2013-05-07	V0.5	韩亚平	根据刘剑意见进行修订
6	2013-05-09	V0.6	韩亚平	去掉关于 git 操作的描述
7	2013-05-11	V0.7	韩亚平	根据谢东总意见修订 1、细化主开发者、开发者管理模式的描述； 2、增加稳定版的分支描述； 3、增加针对分支建立、维护的补充说明 4、删除与规范不相关的工具选择等描述； 5、文字错误修订

目 录

1. 引言.....1

1.1 编写目的.....1
1.2 使用范围.....1
1.3 工具的选择.....1

2. 代码存储策略.....1

2.1 代码库管理策略.....1
2.2 代码分支管理策略.....3
2.3 代码库安全策略.....3
2.3.1 代码库管理：.....3
2.3.2 代码库备份安全管理：.....4
2.4 GIT 主数据库管理.....4
2.4.1 Git 服务器管理员.....4
2.4.2 Git 服务器的统一规则约定：.....4

3. 代码管理过程.....4

3.1 新建项目.....4
3.2 分支建立.....6
3.3 本地代码修改，提交、合并、解决冲突.....6
3.4 代码持续构建、代码持续集成.....7
3.5 产品线集成与发版.....7
3.6 产品维护.....8

4. 基于 GIT 的管理操作过程.....8

4.1 GIT 的基本原理：.....8
4.1.1 版本模型：.....8
4.1.2 Git 文件状态：.....9
4.1.3 Git 管理数据流向图：.....9
4.2 GIT 操作过程.....9

5. 补丁版本管理.....10

5.1 补丁管理现状.....10

1. 引言

1.1 编写目的

为了统一 UAP 研发中心的代码管理规则及工具；统一管理过程及认识。

1.2 使用范围

本规范适用于 UAP 研发中心所有开发产品。

采用逐步推行的方式，在 2013 年完成各产品线的统一管理；做到产品线内部特性分支、开发分支管理；研发中心进行集成分支、发布分支管理。

1.3 工具的选择

1、 选择 Git 作为代码管理工具平台；

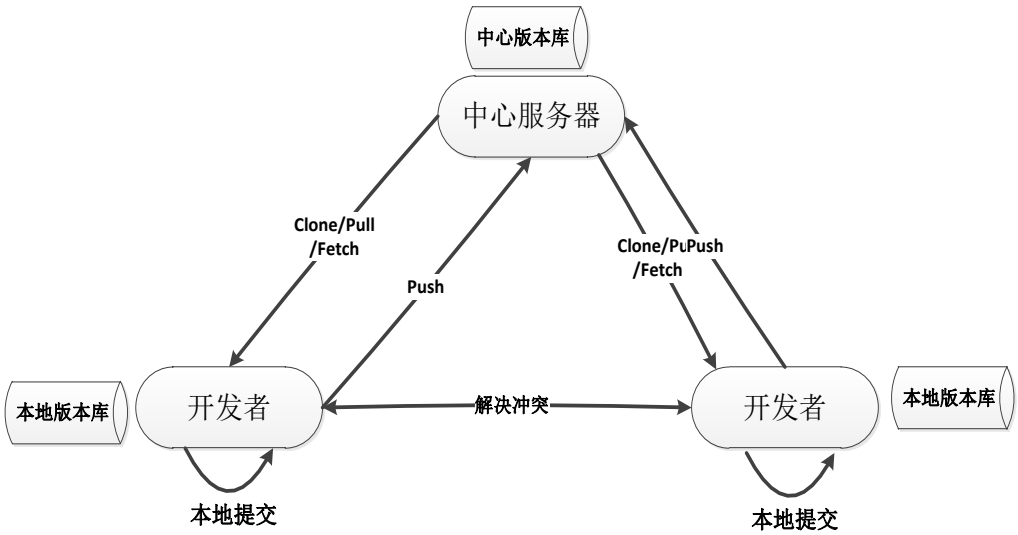
2、 选用 Gitlab 作为服务器端管理工具；

3、 选择 TortoiseGit 作为客户端管理工具；

2. 代码存储策略

2.1 代码库管理策略

Git 为分布式版本控制系统，分布式和集中式的最大区别在于开发者可以本地提交。每个开发者机器上的都有一个和服务器相同的数据库，都是完整的。



开发过程如下：

1、 开发者从服务器上克隆数据库（包括代码和版本信息）到本地机器上；

2、 在本地的机器上创建分支，修改代码；

3、 在本地机器版本库对应分支上提交代码 。

4、 在本地机器上进行与主数据库的代码合并；

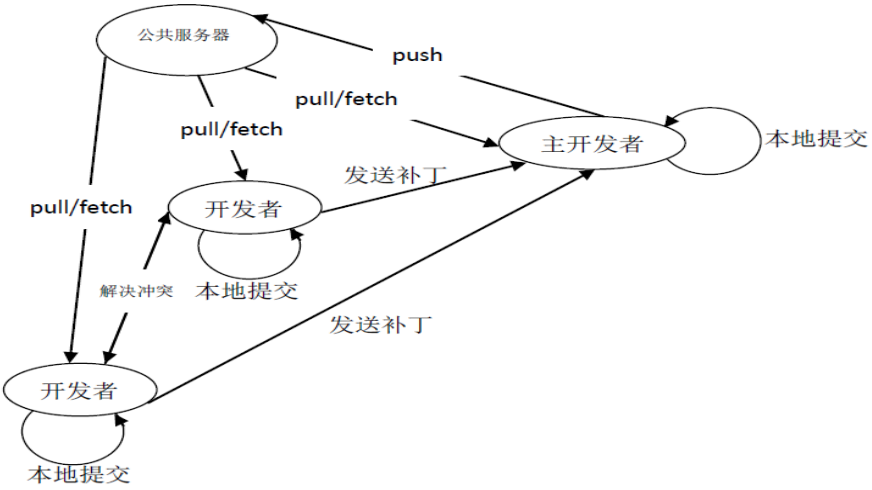
a) 在本地新建立分支，把服务器上最新版的代码 fetch 下来，然后跟自己的主分支合并。

b) 合并过程中如存在冲突，本地解决冲突；

5、 向服务器提交最终修订结果，进入自动构建、持续集成验证。如在 fetch 后长时间没有提交，导致服务器代码有新的提交，可能仍旧提交失败，需要重新 fetch 解决冲突；

6、 开发者也可以在提交到中心库之前，与其他开发者先进行代码的合并和冲突的解决，解决后直接提交至中心数据库中。从而避免影响范围扩大。

由开发人员直接进行主开发分支的操作，根据 Git 的管理模式，通常在大项目中，会设定代码主开发者，对主版本库进行独立维护和管理，解决主版本冲突等问题，各开发者将改动统一提交至主开发者，这样也保证了对主数据库操作的稳定性，如下图



目前我们各产品线可以根据团队的成熟状态，来确定是否这顶主开发者的角色，我们建议采用混合的方式进行代码的管理，即：

1、 团队内设定经验丰富，代码开发质量的成员为主开发者，具有向中心主数据库提交代码的权限；对将要提交至主数据库的代码进行 Code Review，从而保证提交到主数据库代码质量。

2、 设定新员工或初级程序员为普通开发者，普通开发者从主数据库获取代码，但向主开发者提交代码；不能直接提交至主数据库。

3、 根据团队成熟度主开发者可以有数个，根据团队代码开发的质量，人员的能力提升，团队的成熟，普通开发者逐步升级为主开发者，从而保证团队的良性循环。

尤其在 Git 推行的初期阶段，建议设定主开发者，并通过此角色来保证代码的质量。

2.2 代码分支管理策略

Git 鼓励通过分支进行管理，以方便日常开发，我们根据分支的目的开发，分为如下代码分支：（见下页表格）

多产品管理： 每个产品维护自己独立的代码库，代码库以产品名称命名，内部分支各产品库统一命名，

分支类别	作用说明	提交要求	分支来源	创建人	命名规则	分支位置	分支合并	备注
特性分支 feature branch	开发中，用于新需求的实现；由开发人员创建，作为开发人员日常开发的基础分支	开发人员掌控	develop branch	开发人员	账号/分支特性名（或特性编号） 如：liujian/add_new_dialog（WFP-599）	一般只存在于各开发的本地库，不 push 到主数据库中		
开发分支 develop branch	开发中，某一产品集成分支，为开发人员日常开发的分支，develop 是每日构建的代码来源，产品提交产品线集成后， develop 分支同时可以继续做其他特性的开发	完成开发人员本地验证，通过单元测试	Gold branch 或初始创建	开发经理	单一命名为 Develop	主数据库		
待发布分支 release branch	产品线集成后建立，从 develop 分支分出，产品部门当开发到一定的阶段进入固化状态时，提交至产品线进行集成时创建待发布分支，该分支只做 Bug 的修订	完成单产品集成，达到单产品质量标准	develop branch	配置管理员	R+版本号+集成日期，如：RV1.0-130309，代表 1.0 版本 13 年 3 月 9 日提交产品线集成盘对应代码分支；	主数据库	release 分支上的内容需随时合并 到 develop 分支，要求在 BUG 修订后关闭的当天合并	
金盘发布分支 Gold branch	发版时建立，当产品验证到了稳定的状态,即发版的质量标准后，把 release 分支合并到 Gold 分支，也就是发布了新版本	完成产品线集成，达到产品发版质量标准	release branch	配置管理员	单一命名 Gold	主数据库	BUG 修订后在 BUG 关闭的当天必须合并回 develop 分支以保证 develop 分支的完整性	测试经理发部质量报告，经过发版申请后，
补丁分支 hotfix branch	维护阶段建立，补丁分支主要用来修改已发布版本的 bug	Bug 修订完毕，通过开发自我验证	Gold branch	开发经理	单一命名为 Hotfix Branch	主数据库	补丁发布后，需要将补丁修改内容，即使合并至当前主开发分支	

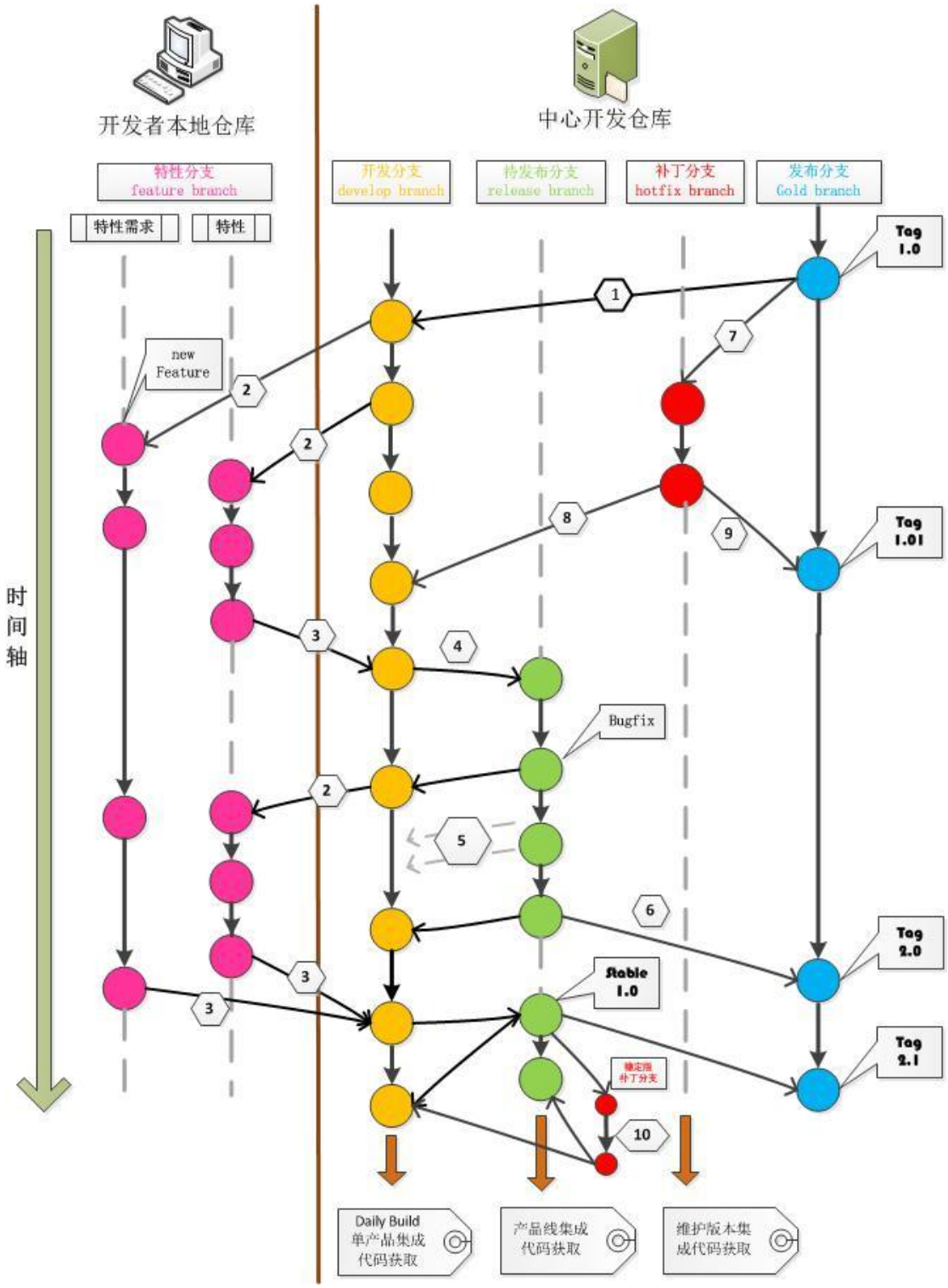
针对分支的补充说明：

1、 开发经理提交产品线集成申请，测试经理进行测试接受性验证后，如通过，则通知配置管理员，配置管理员建立 release 分支。

2、 产品线持续集成分为个阶段：一个为产品线持续集成 BUG 修订阶段，一个为金盘验证阶段，当产品线集成阶段进入第二阶段后，代码进入冻结状态，配置管理员收回产品线持续集成分支的权限，仅保留配置管理员及开发经理的写权限，对于代码的修订控制在最小，除非必要修订的 BUG，不再做其他修改。

3、 稳定版发版：当产品线持续集成阶段进入到稳定阶段，可以对外发布一个相对稳定的版本时，配置管理员在经过测试经理稳定版发布通知后，在产品线持续集成分支进行 tag 标识。并在此标识版本进行稳定版的构建（或构件完成后，配置管理员找到对应版本进行标识），稳定版本原则上不进行补丁的修订，本次稳定版反馈的问题待下一稳定版本解决，出现特殊情况，必须进行修订，则建立临时稳定版补丁分支，进行临时修订，通过补丁的形式发布到客户，同时本次补丁修订内容需及时合并至产品线集成分支及开发分支；

各分支结构关系图如下：



说明：

1 ： 建立产品开发分支，从某个前期发布的金盘版本建立或从零开始建立初始代码库；

2 ： 开发人员在本地库根据开发任务建立特性分支或缺陷分支，作为独立开发的环境；以特性或缺陷为单位进行独立完整的开发及提交。

3： 开发人员完成自我验证，提交到开发主分支（集成分支）执行单产品集成；从这一点开始，开发人员进入新特性开发及 BUGfix 的并行阶段；

4：单产品集成达到稳定版本，提交到待发布分支（产品线集成分支），开始执行新版本的产品线集成；

5：在待发布分支上进行的 Bug 修订，需要持续的 Merge 回开发主干开发分支中，从而后续主开发版本的完整性和质量；

6：产品线集成完毕，达到一定质量标准，进行发版，将对应代码 Merge 至金盘分支，同时进行 tag 标识。

7：在发版后建立维护分支，执行发版产品的缺陷修订，解决客户问题；

8：持续将补丁分支的修订代码 Merge 至主干开发分支，以保证主开发版本的完整性和质量。

9： 补丁版本发布，提交至金盘分支；同时进行 tag 标识；

10：建立临时稳定版补丁版本：

2.3 代码库安全策略

2.3.1 代码库管理：

阶段	开发经理	UAP 配置管理员
代码库 初始建立	确定明确如下事项后，向配置管理员提出建库申请： 1 产品命名 2 产品内项目的分布； 3、确定代码库的访问人员， 4、各项目的人员访问权限， 5、负责代码库人员及权限维护	根据开发经理的申请 1、 在代码服务器建立产品 group 即产品的 project； 2、 建立产品 Team 及 user，并赋予相应的访问权限； 3、 建立并管理用户及密钥 4、 管理用户组和用户之间的关系
日常 分支管理	1、负责特性分支的管理 2、负责 develop 分支的管理	1、负责集成分支管理 2、负责发布金盘分支的管理；

	3、负责提出产品线集成申请，进行集成分支代码合并	

2.3.2 代码库备份安全管理：

代码库的备份管理由集团开发管理部统一进行；

2.4 Git 主数据库管理

2.4.1 Git 服务器管理员

设定 Git 主数据库服务器管理员，负责 Git 主数据服务器的维护。主要包括：

1、 Git 服务器 SSH 用户及用户组的规划、设计及建立。
2、 新产品主数据库的建立及维护；
3、 新产品主数据库的用户及权限设定；
4、 Git 服务器的日常维护；
5、 Git 服务器操作过程中异常处理；

2.4.2 Git 服务器的统一规则约定：

1、以产品为单位建立 group；
2、开发经理确定项目的建立规则；

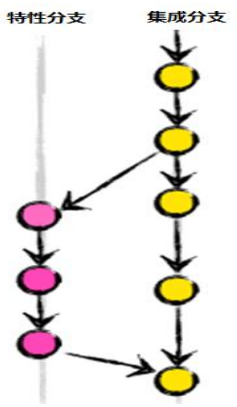
3. 代码管理过程

3.1 新建项目

1、 新产品启动后，首先由开发经理在中心服务器建立本产品的初始 develop 库；如在已经发版基础上进行二次开发，则以发布分支为基础，建立开发分支；
2、 开发人员从服务器上获取基准数据库，有两种取得 Git 项目仓库的方法。第一种是在现存的目录下，建立新库，通过导入所有文

件来创建新的 Git 仓库。第二种是从已有的 Git 仓库克隆出一个新的镜像仓库来（包括代码和版本信息）到本地机器上；

3.2 分支建立



1、 开发人员在本地开发过程中，以特性（或缺陷）为单位展开代码开发工作，即根据特性（缺陷）为独立单位建立特性代码分支，在特性分支上完成独立代码开发的功能；特性分支从开发分支初始建立（缺陷分支从发布分支初始建立）；

2、 原则上我们在主服务器仅管理主开发（Develop）分支代码即集成分支，针对特性分支仅保留在开发人员本地代码库中。

3.3 本地代码修改，提交、合并、解决冲突

1、 开发人员在本地特性分支进行代码修改，根据开发内容的不同，在不同的特性分支之间进行切换，从而保证更改是独立的，且在一个干净的环境中开始和结束；

2、 独立的特性(或缺陷)完成并独立验证后，合并至本地的主干分支上（即本地 master）

3、 分支合并时，要求开启--no-off 选项，以保留特性分支的痕迹，方便日后区分很哪些提交是在一起是为了实现一个特性

4、 开发人员在本地验证测试通过后，将代码提交至主服务器的集成分支，要求提交代码为具有质量保证自测无问题代码；

5、 开发人员在提交本地代码前，首先需要将服务器集成分支的最新代码 fetch 到本地，进行本地代码合并及冲突的解决。冲突解决后代码方可提交至集成分支。

6、 开发人员原则上要求每天做到本地代码库提交，每三天至少进行一次集成分支的合并提交，提交尽可能以完整功能为单位，如规定时间内的成果无法做到完整提交，则需要尽快提交，不能大于一周，从而避免产生大量的代码冲突；

3.4 代码持续构建、代码持续集成

1、 开发人员在每天规定时间前完成特性（及缺陷）的代码提交，将代码由本地分支提交至产品集成分支（Develop branch）；该时间后提交特性将只能纳入第二日的构建中。

2、 每日在约定的时间系统自动开始执行产品分支的自动集成构建；

a) 构建失败，系统发出邮件给相关人员，开发人员在第二日接收到邮件后，首先进行构建问题的修改，从而保证当日构建的成功；

b) 构建成功后，执行单元测试代码及自动化测试脚本，将出现的 BUG 通过系统自动发送给相关人员；

c) 存在缺陷的人员，进行缺陷修订，在第二天提交正确的代码；

3、 开发人员首先要保证提交至集成分支代码的稳定构建运行，从而不影响其他人员的开发，其次为进行新特性分支的开发；

3.5 产品线集成与发版

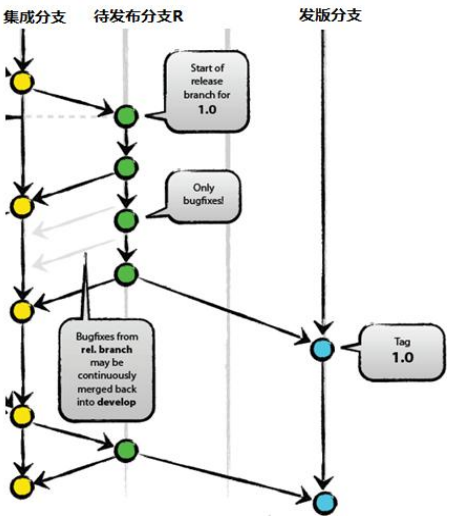
1、 当部门内完成内部集成，达到一个相对稳定版本后，则提交产品线集成；

2、 此时由开发经理建立待发布分支（release 分支），产品线集成版本来源于待发布分支，集成过程中发现缺陷，在此分支上进行修改。

3、 开发人员 pull 本分支到本地，建立缺陷分支，完成缺陷的代码修改合并；

4、 开发人员可以继续在 develop 分支上进行新特性的开发。

5、 在待发布分支上进行的缺陷修复，在缺陷关闭后，需要实时



将更改同步到 develop 分支上，从而避免后期产生大量的冲突，

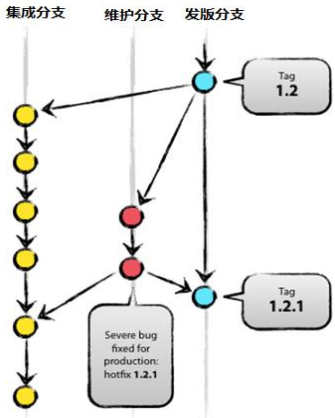
6、 开发经理负责定期检查本产品的分支合并情况。避免后期出现代码合并冲突风险。

7、 产品经过产品线集成测试后，进入发版环节，此时在待发版分支建立发布分支，表明金盘对应的版本；

3.6 产品维护

1、 发版后，进入维护阶段，产品在使用过程中发现的问题，通过建立维护分支实现对问题的修订；

2、 开发经理同样需要关注维护分支与开发集成分支的代码同步，定期进行合并。

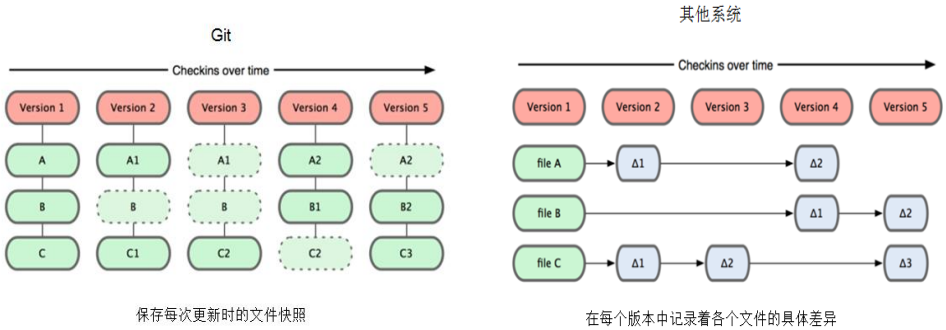


4. 基于 Git 的管理操作过程

4.1 Git 的基本原理：

4.1.1 版本模型：

1. 直接纪录仓库的一个快照，而非文件的差异比较

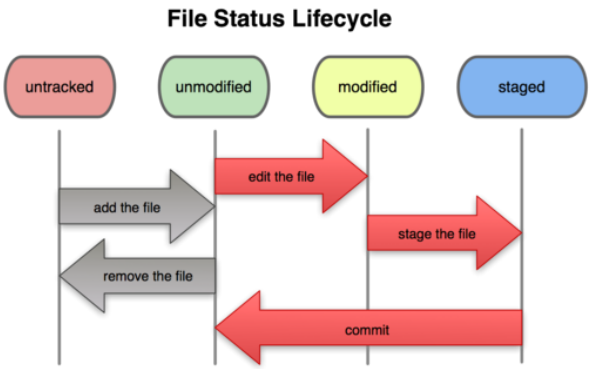


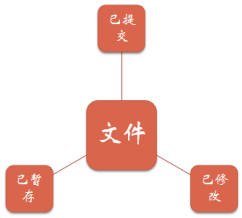
2. 近乎所有操作都是本地执行，绝大多数操作都只需要访问本地文件和资源，不用连网；
3. 时刻保持数据完整性：在保存到 Git 之前，所有数据都要进行内容的校验和（checksum）计算，并将此结果作为数据的唯一标识和索引，所

有保存在 Git 数据库中的东西都是用此哈希值来作索引的，而不是靠文件名

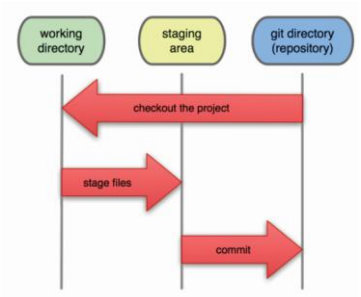
4.1.2 Git 文件状态：

1. 文件的三种状态：
   a) 已修改：修改了某个文件，但还没有保存
   b) 已暂存：把已修改的文件放在下次提交时要保存的清单中
   c) 已提交：已被安全地保存到本地数据库中
2. 文件状态周期图





3. 基本的 Git 工作流程
   a) 从 git 仓库中 checkout 项目 到工作目录。
   b) 在工作目录修改某些文件。
   c) 对修改后的文件进行快照，然后保存到暂存区域。(Add to Staging Area (Index))
   d) 提交更新（commit），将保存在暂存区域的，文件快照永久转储到 Git 仓库 (Repository 中
   e) Git 的文件流转的三个工作区域工作目录、暂存区域（也叫索引文件）、Git 本地仓库



4.1.3 Git 管理数据流向图：

《待补充》

4.2 Git 操作过程

1. Git 目录
   a) |-- HEAD　　# 这个 git 项目当前处在哪个分支里

b) |-- config　　# 项目的配置信息，git config 命令会改动它

c) |-- description　# 项目的描述信息

d) |-- hooks/　　# 系统默认钩子脚本目录

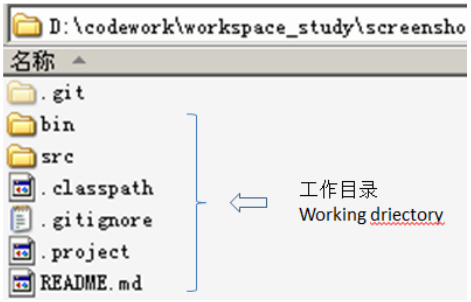
e) |-- index　　# 索引文件

f) |-- logs/　　# 各个 refs 的历史信息

g) |-- objects/ # Git 本地仓库的所有对象 (commits, trees, blobs, tags)

h) |-- refs/　# 标识你项目里的每个分支指向了哪个提交(commit)。

2. Git 本地仓库和工作目录



5. 补丁版本管理

《待补充》

5.1 补丁管理现状

- 发版后，开发流就锁住，启用一个 patch 流
- ClearCase 无法很好地处理多分支的专项补丁
- 补丁只能 Hijack 修改，无法集中备份
- 无法统一管理补丁源码
- 各人的管理方式都不一样
- 工作交接是问题
- 补丁冲突只能在现场发现
- 合并补丁工作量巨大(比如 v575)
- 开发补丁的准备工作时间很长