FIT2CLOUD 飞致云

高校合作计划 FIT2CLOUD 飞致云

基于 GitHub/Gitee 完成开源协作

FIT2CLOUD 飞致云 培训认证中心

2024 年 11 月

FIT2CLOUD 飞致云

目录

CONTENTS

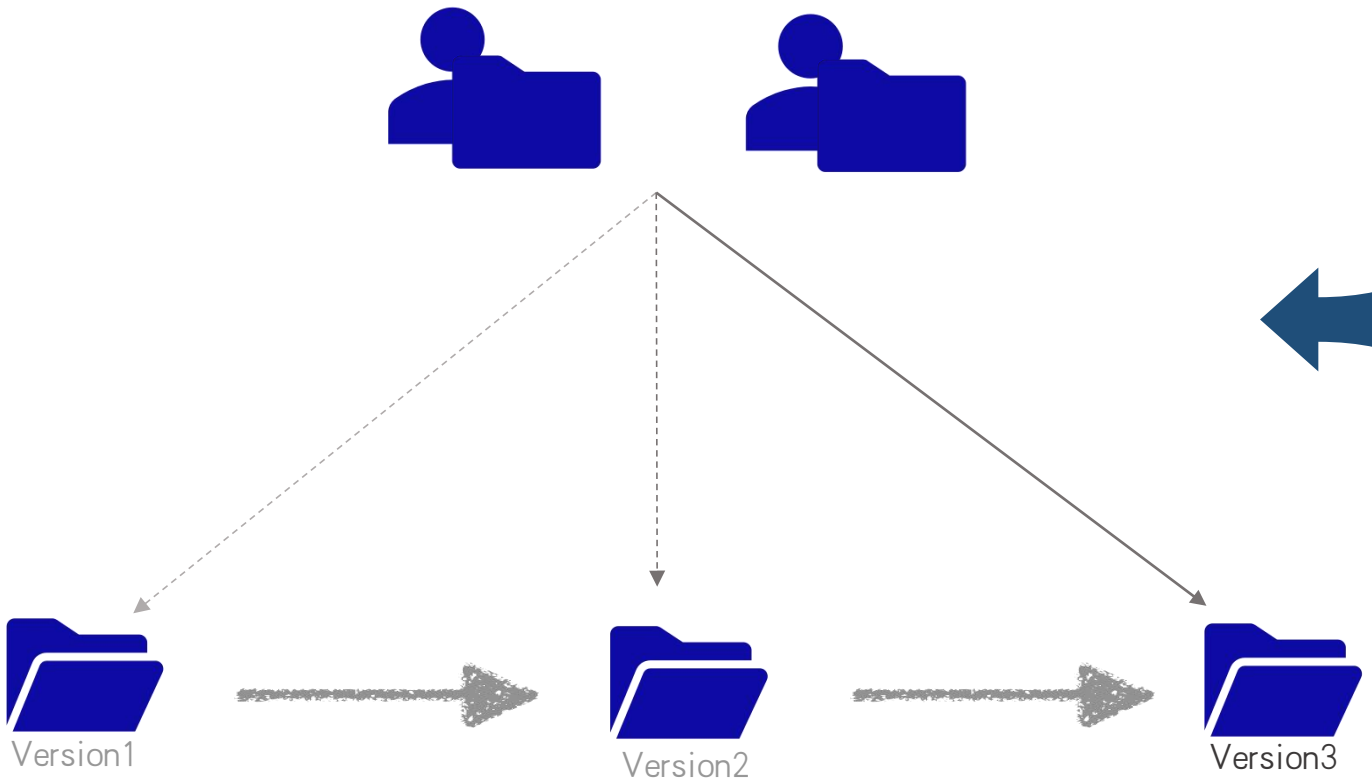
01

版本控制

- 概念介绍
- 阶段演变
- 系统比较
- 价值好处

什么是版本控制?

版本控制是一种记录一个或若干文件内容变化，以便将来查阅特定版本修订情况的系统



管理文件、代码等内容的修改历史，追踪文件的变更、记录更改历史记录、备份以便恢复以前的版本

管理

协同

多人协同作业，版本控制可以有效地解决版本的同步以及不同开发者之间的开发通信问题，提高协同开发的效率

为什么需要版本控制系统?

– 文件操作的时候可能遇到的几种情况 –

- 当两个开发人员进行合作开发，操作同一个文件时，需要把文件内容合并的话，需要手动拷贝代码到同一个文件中!
- 小团队合作开发时，新开发的代码文件需要通过U盘互相拷贝，可能高级一点的会开个文件共享服务器，自己下载后合并!

U盘拷贝

- 当不小心手残把文件删除掉，或者修改掉，想找回修改前的文件内容，却不知道如何下手!
- 当代码不小心被团队某个人删掉，并且没有做任何的备份!

版本控制系统比较

本地版本控制系统

- 本地版本控制系统是位于本地计算机上的本地数据库，其中每个文件更改都作为补丁存储。每个补丁集仅包含自上一版本以来对该文件所做的更改。
- 主要问题是所有内容都存储在本地。如果本地数据库出现任何问题，所有补丁都将丢失。如果单个版本出现任何问题，则该版本之后所做的所有更改都将丢失。开发人员或团队协作时非常困难或几乎不可能。
- 代表系统：源代码控制系统(SCCS) — 1972、修订控制系统(RCS) — 1982。

集中式版本控制系统

- 集中式版本控制系统有一个包含所有文件版本的服务器。这使多个客户端可以同时访问服务器上的文件，将它们拉到本地计算机或将它们从本地计算机推送到服务器。这允许与其他开发人员或团队轻松协作。
- 最大问题是一切都存储在中央服务器上。如果该服务器发生问题，则没有人可以保存他们的版本更改、拉取文件或进行任何协作。
- 代表系统：并发版本控制系统(CVS) — 1986、ClearCase — 1992、Subvison（SVN）— 2000、Microsoft Team Foundation Server (TFS)

分布式版本控制系统

- 分布式版本控制系统，客户端不仅可以从服务器上检出文件的最新快照，还可以完全镜像存储库，包括其完整历史记录。因此，在一个项目上合作的每个人都拥有整个项目的本地副本，即拥有他们自己的本地数据库以及他们自己的完整历史。使用此模型，如果服务器变得不可用或死机，任何客户端存储库都可以将项目版本的副本发送到任何其他客户端或在服务器可用时返回到服务器。
- 代表系统：Git，衍生的GitHub、Gitee、Gitea、GitLab等。

版本控制的演变

本地版本控制

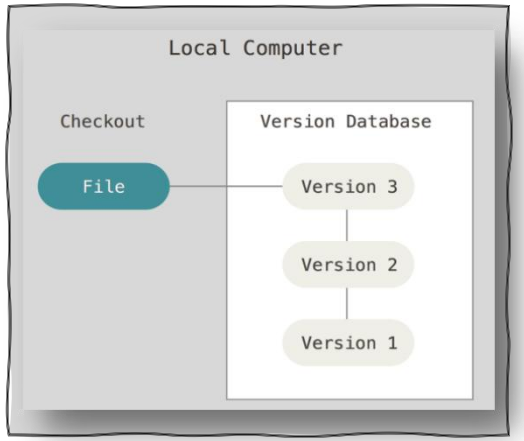
复制整个项目文件，并以时间对文件夹进行命名加以区分文件版本

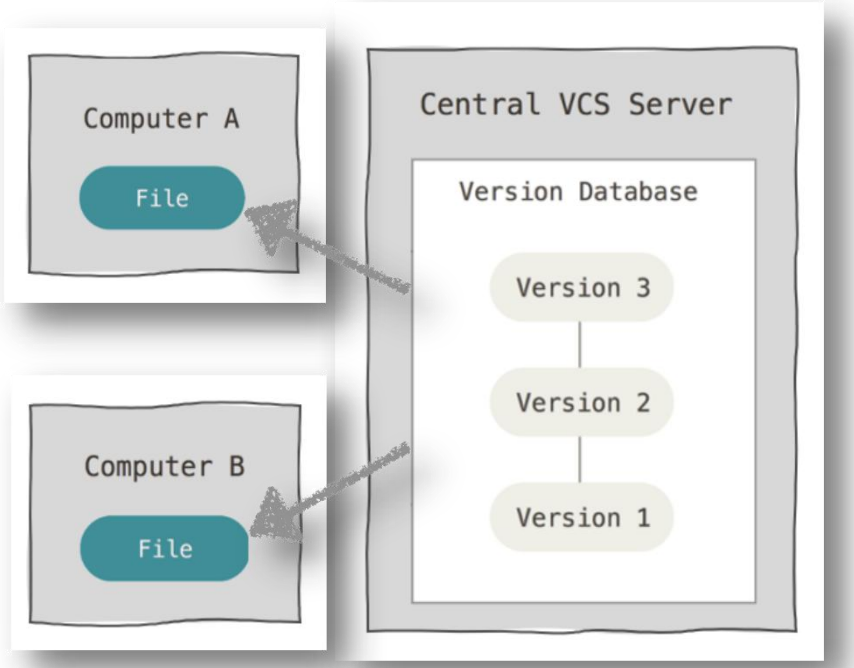
集中式版本控制

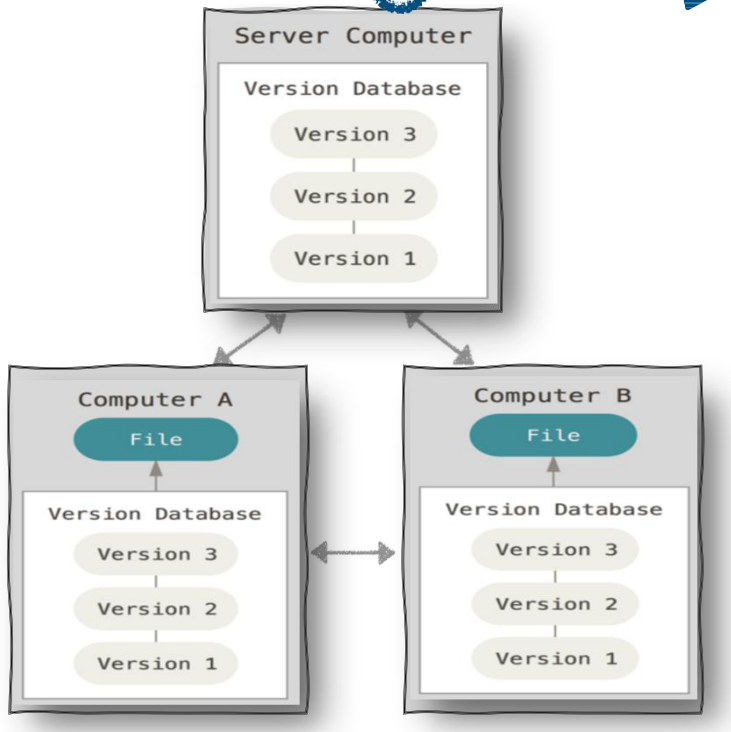
单一的集中管理的服务器，保存所有文件的修订版本，而协同工作的人们都通过客户端连到这台服务器，取出最新的文件或者提交更新

分布式版本控制

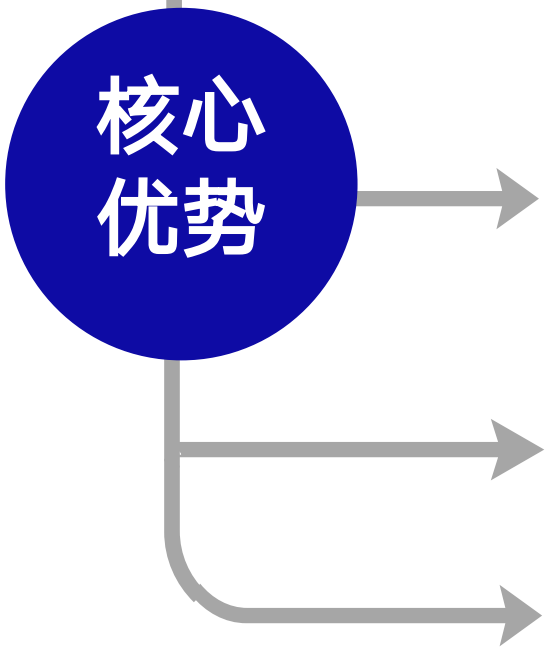
没有明确的中央服务器的概念，每一个客户端都是一个完整的版本库。任何一处协同工作用的资产发生故障，都可以用任何一个镜像出来的本地仓库恢复。







版本控制系统的核心优势



每个文件的完整长期更改历史

每个文件的每一个改变，包括创建、删除、编辑，同时也会记录作者、日期和没次更改目的的书面说明，拥有完整的历史记录可以回溯，分析bug根本原因。

快速合并和灵活分支

不需要远程服务器通信，所有代码可以快速合并，还允许团队使用不同的分支策略，让开发阮对专注创新，快速构建。

快速反馈和较少的合并冲突

因为在本地工作站上拥有整个存储库的历史可以确保开发人员可以快速地进行实验并请求代码审查，开发人员可以从快速反馈循环中受益，并且可以在合并更改集之前与团队成员共享更改 。

离线工作的灵活性

分布式版本控制系统不需要互联网连接，所以除了推拉之外，大多数开发都可以在旅行或远离家或办公室的时候完成 。

可追溯性

能够跟踪对软件所做的每个更改，并将其连接到项目管理和错误跟踪软件(如Jira)，并且能够用描述更改的目的和意图的消息注释每个更改，不仅可以帮助进行根本原因分析和其他验证 。

工作原理

- Git是什么
- Git的特性
- Git工作原理
- Git基础命令

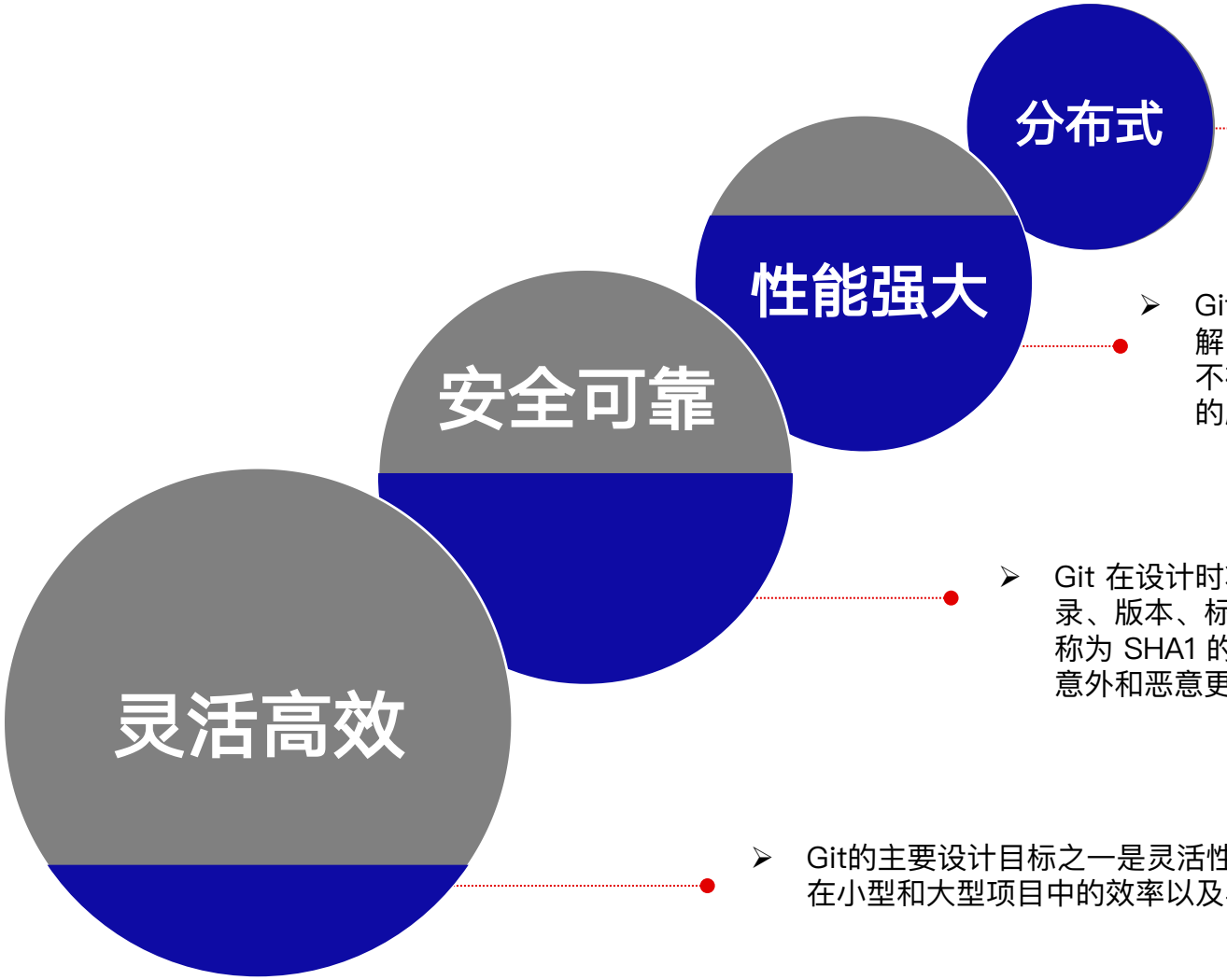
什么是 Git?

– Git是一个开源的分布式版本控制系统，使用Git可以免费的、高效率的进行代码托管，这也是目前Git较为流行的原因 –

Git的优势
提交完全在本地完成，无须别人给你授权，你的版本库你作主
每次提交都相当于一个全备份，加快版本切换的速度
Git可以从任何一个版本库的克隆来创建属于自己的版本库
允许不同人员在不影响主代码库的情况下进行实验性更改

- 可离线完成大部分的操作，不受网络和中央服务器影响
- 开源开放，效率更高
- 更强大的分支拉取和合并操作
- 更强的撤销修改和修改版本历史的能力

Git 的设计特性



- 分布式解决了本地版本控制以及集中式版本控制存在的问题。
- Git 的内部实现算法基于对真实源代码文件树共同属性的深入了解，在确定文件树的存储和版本历史时，专注于文件内容本身，不被文件名迷惑。所以在提交新的更改、分支、合并和比较过去的版本在性能方面都有非常强大的性能提升。
- Git 在设计时将托管源代码的完整性作为首要任务。文件的内容以及文件和目录、版本、标签和提交之间的真实关系，Git 存储库中的所有这些对象都使用称为 SHA1 的加密安全哈希算法进行保护。这可以保护代码和更改历史免受意外和恶意更改，并确保历史完全可追溯。
- Git的主要设计目标之一是灵活性。Git在几个方面都很灵活：支持各种非线性开发工作流程，在小型和大型项目中的效率以及与许多现有系统和协议的兼容性。

Git 工作原理

01 Git的整体体系架构

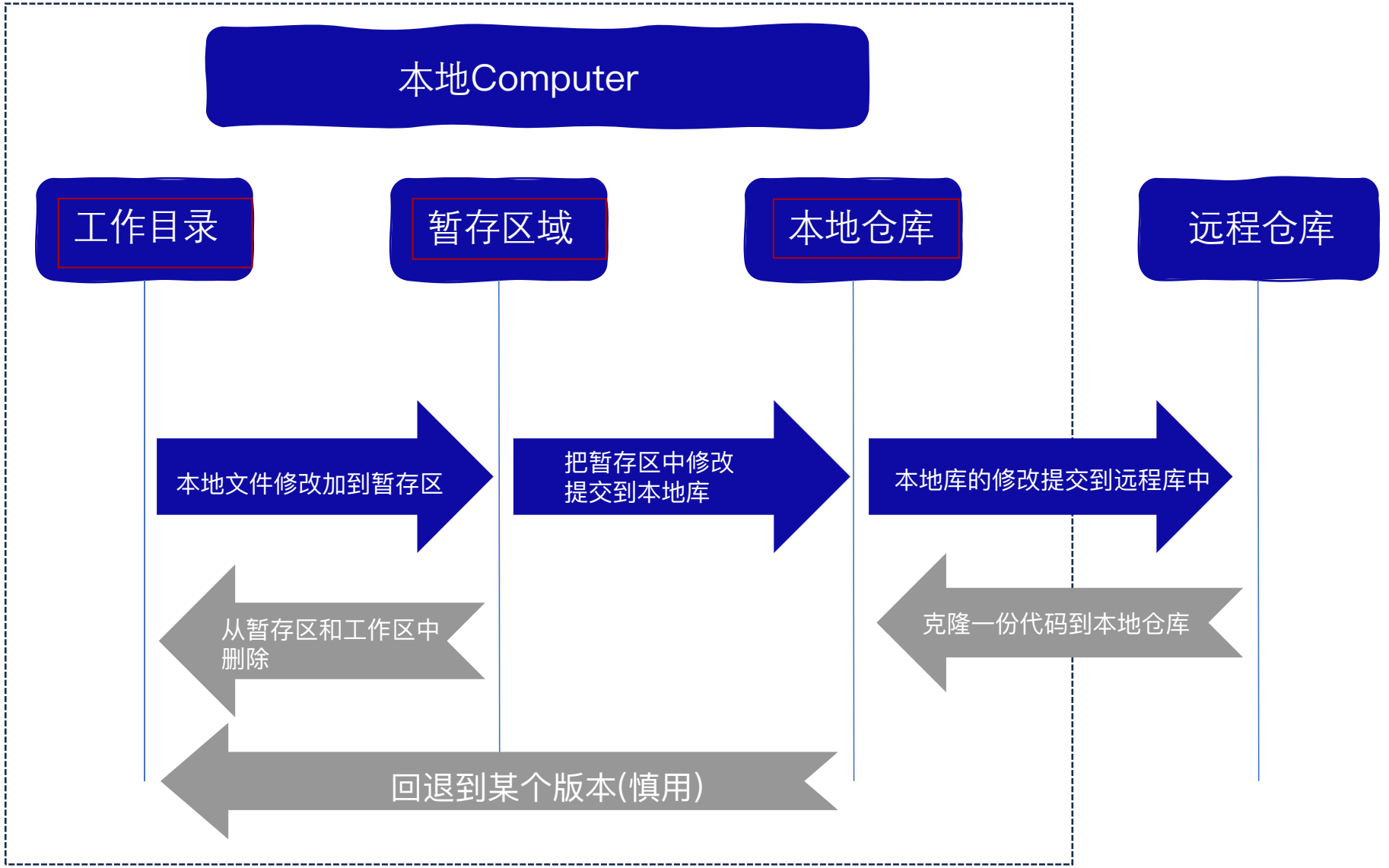
02 Git的工作流程

03 Git的核心概念

Git 整体体系架构

名词介绍

- 工作目录：开发进行写代码、修改代码的地方
- 暂存区域：在提交进入本地仓库之前，可以把所有的更新放在暂存区
- 本地仓库：一个存放在本地的版本库，包含了项目的完整历史记录
- 远程仓库：在Git远程服务器的一个文件目录，一般使用Github、Gitlab和Gitea。

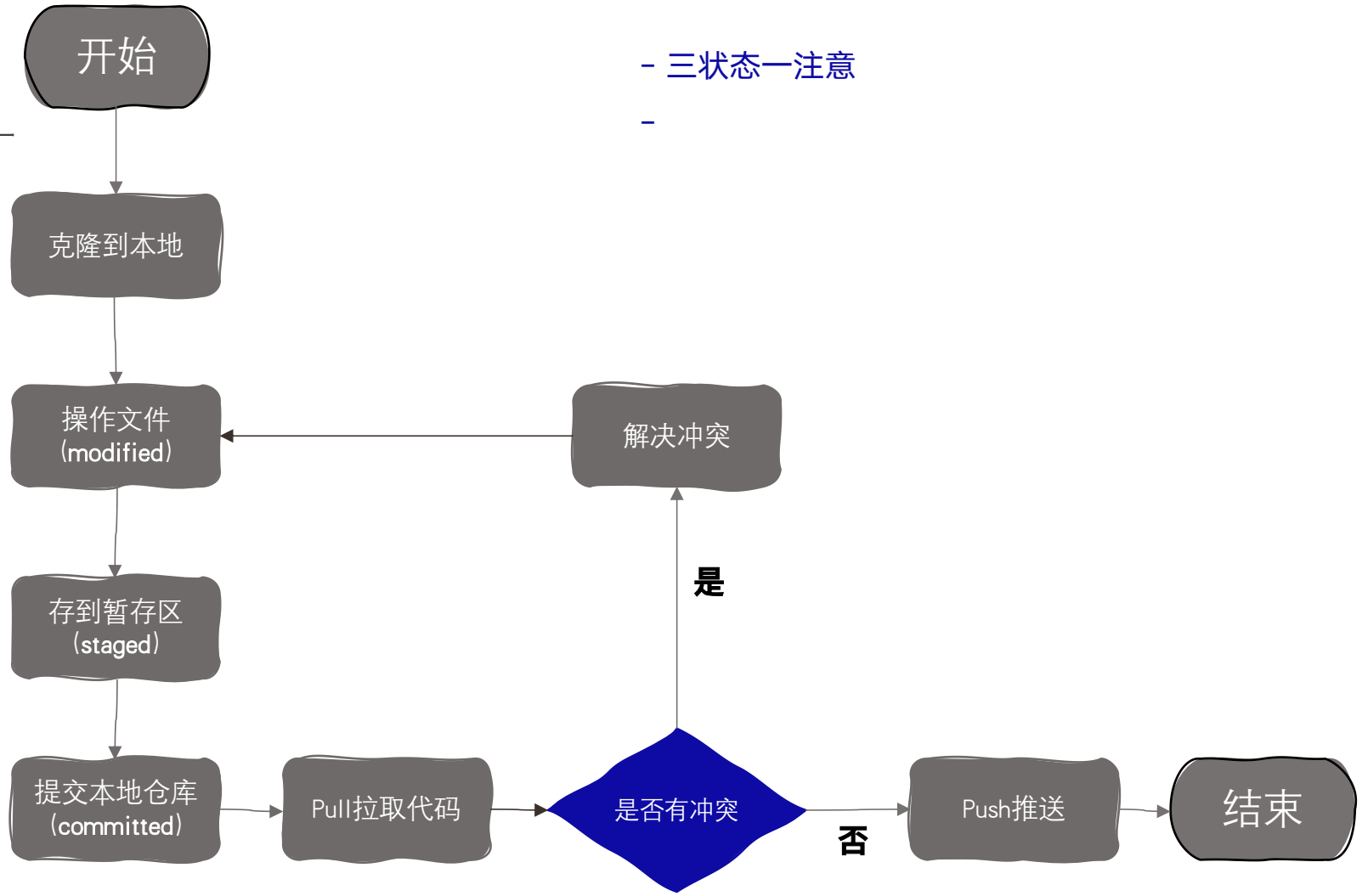


Git 工作流程

Git有三种状态，文件可能处于其中之一

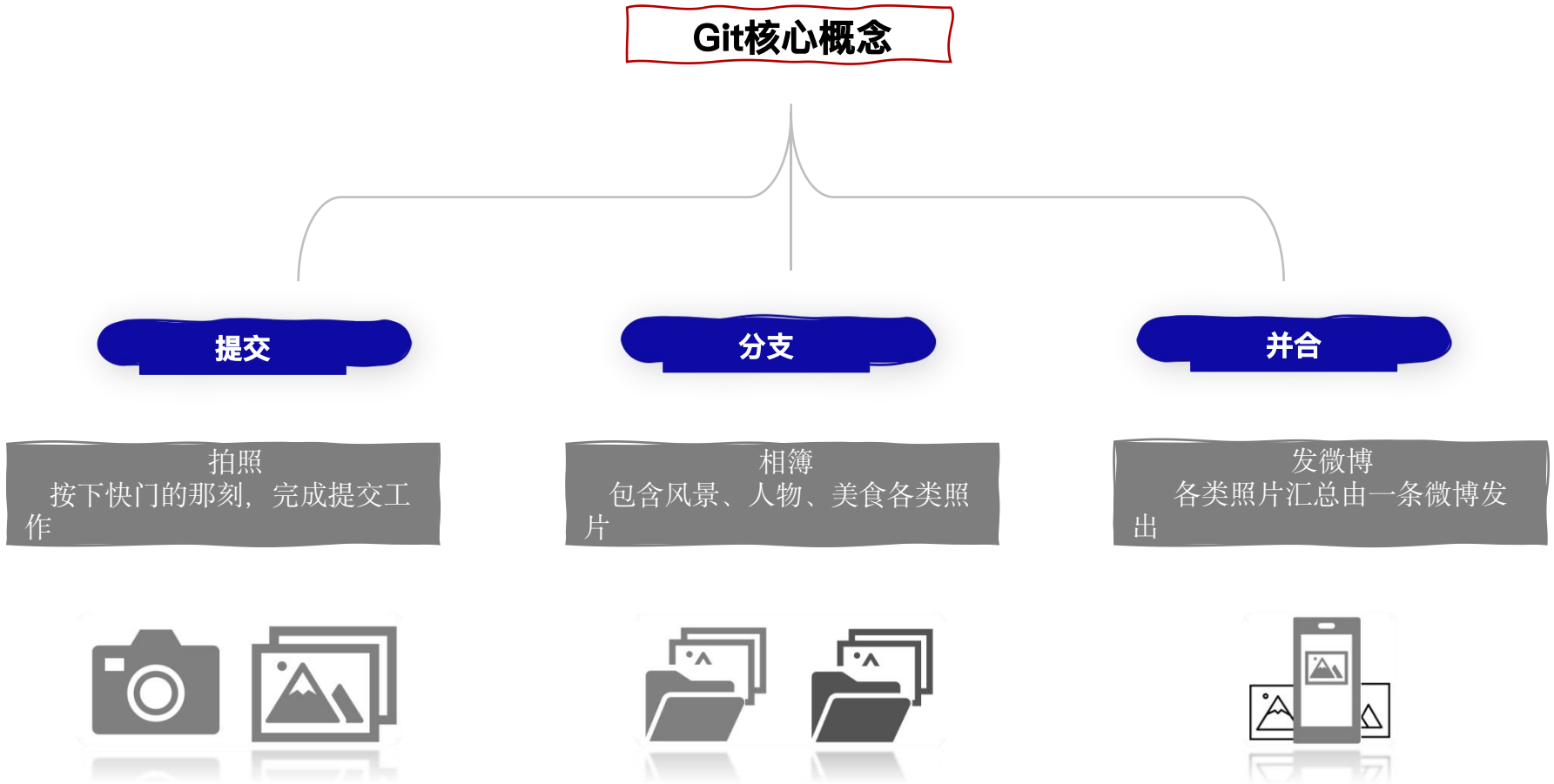
- 三状态一注意
-

- Modified：已修改表示修改了文件，但还没保存到仓库中。
- Staged：已暂存表示对一个已修改文件的当前版本做了标记，使之包含在下次提交的快照中。
- Committed：已提交表示数据已经安全地保存在本地仓库中。
- 修改推送到远程仓库前，一定要先Pull，检查是否有冲突，避免冲掉别人修改。



Git 核心概念

– Git的三大支柱，共同确保代码的完整性和可追溯性，支持高效的团队协作



Git提交

Git对待数据更像是一个快照流

提交过程

每当提交更新或保存项目状态时，基本上就会对当时的全部文件创建一个快照并保存这个快照的索引，包含一个指向上次提交对象的指针。为了效率，如果文件没有修改，Git不再重新存储该文件，而是只保留一个链接指向之前存储的文件。

Git分支

– Git建分支是非常灵活的，可以任意建立分支，对任意分支再分支，分支开发完后再合并

–

查看分支

```
[root@jenkins git_data]# git branch -a
* dev
  master
  remotes/origin/HEAD -> origin/master
  remotes/origin/dev
  remotes/origin/master
[root@jenkins git_data]#
```

切换分支

```
[root@jenkins git_data]# git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
[root@jenkins git_data]#
```

创建分支

```
[root@jenkins git_data]# git branch release
[root@jenkins git_data]# git branch -a
* dev
  master
  release
  remotes/origin/HEAD -> origin/master
  remotes/origin/dev
  remotes/origin/master
```

=

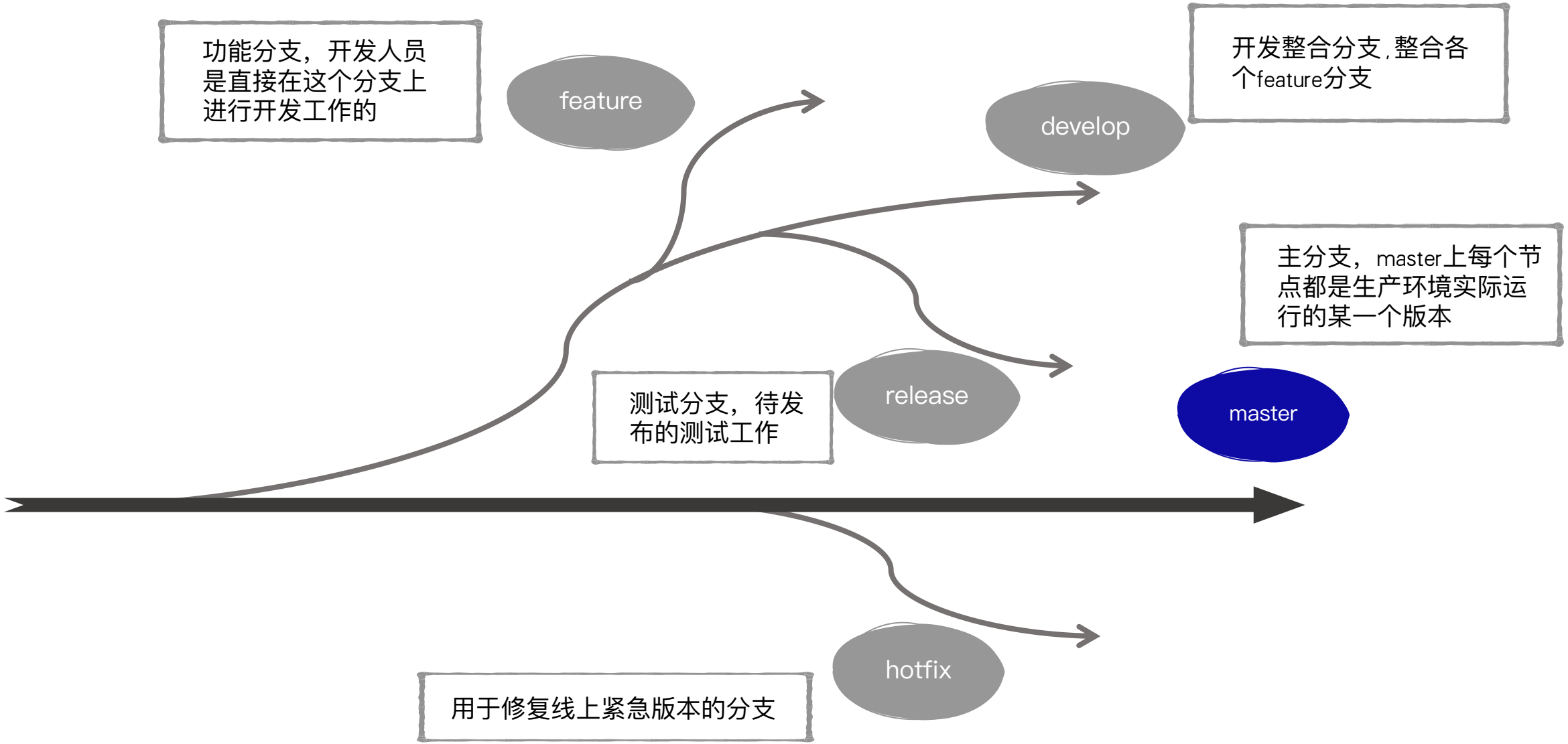
Git分支

使用分支意味着可以把工作从开发主线上分离开来，以免影响开发主线

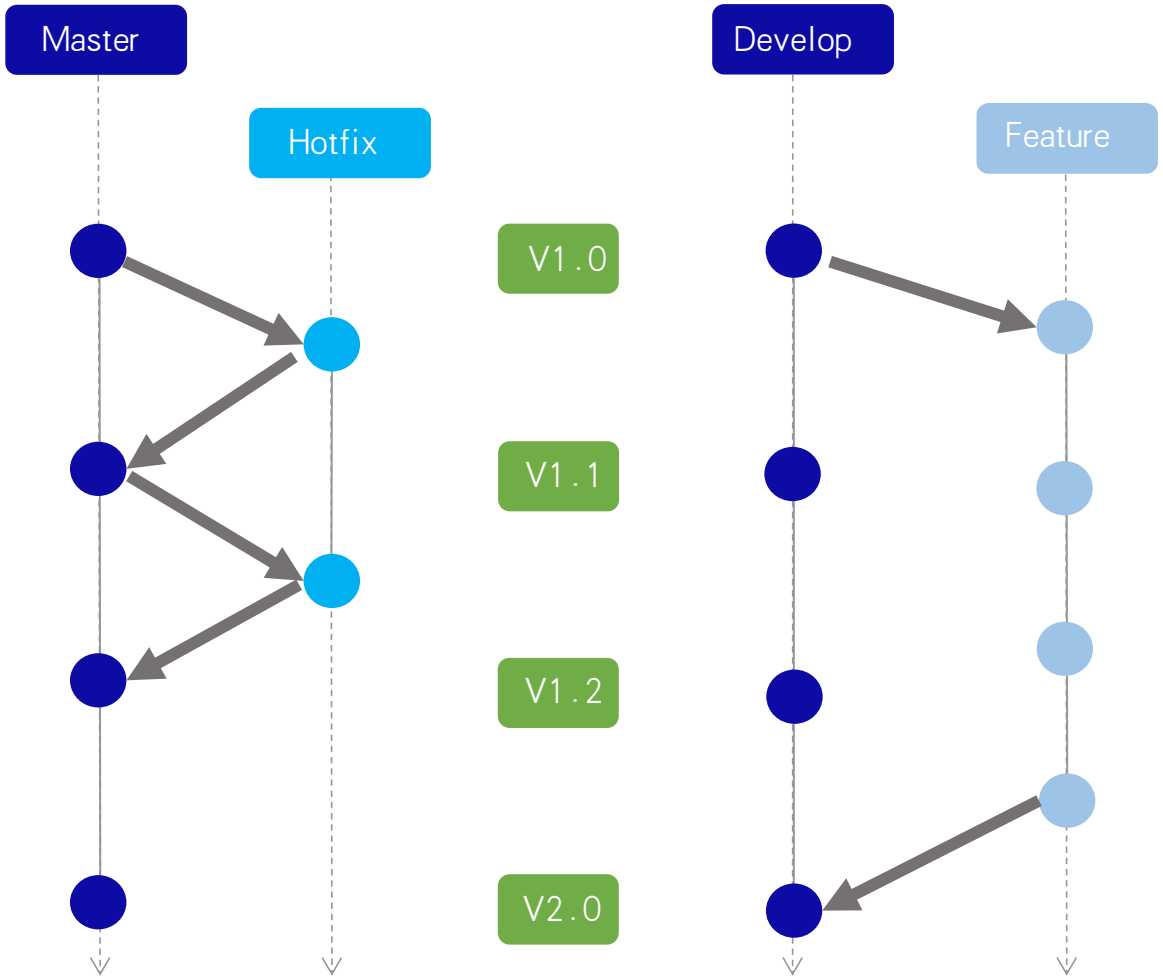
必杀技特性

- 多分支隔离
- 并行开发
- 高效协作
- 分支切换便捷
- 快速尝试和迭代

Git分支管理


功能分支，开发人员是直接在这个分支上进行开发工作的
feature
开发整合分支，整合各个feature分支
develop
主分支，master上每个节点都是生产环境实际运行的某一个版本
测试分支，待发布的测试工作
release
master
hotfix
用于修复线上紧急版本的分支

Git合并管理



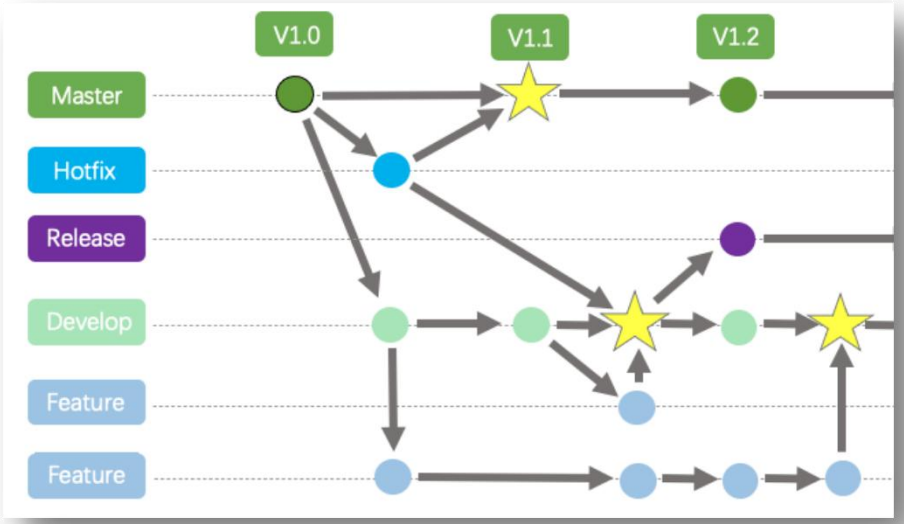

```
[root@jenkins git_data]# git merge dev
Updating b0645b9..cc854ac
Fast-forward
 haha.txt | 1 +
 hello    | 2 ++
 2 files changed, 3 insertions(+)
 create mode 100644 haha.txt
 create mode 100644 hello
```

在合并操作中，Git会先比较两个分支的提交记录，即分支的版本记录。合并操作可以由三个阶段组成：合并前准备、合并以及解决冲突

Git冲突管理

无冲突合并：

指针的移动，不会产生新的commit，Git将合并的结果做了一个新的快照并且自动创建一个新的提交指向它。



有冲突合并：

需要手动解决，Git 会暂停下来，并在文件中标记出冲突的部分。 解决冲突之后，解决完冲突后，Git会自动进行合并提交，将两个分支的代码合并在一个新的提交中。

手动解决冲突，删除不需要的冲突代码

借助工具合并（IDEA等）

使用Git命令，强制覆盖（慎用）

Git实践

- Git的安装
- Git基础命令

Git的安装

- Linux操作系统：https://git-scm.com/download/linux
- Windows操作系统：https://git-scm.com/download/win
- Mac OS操作系统：https://git-scm.com/download/mac
- 本机配置GitHub信息：

  https://blog.csdn.net/weixin_43983960/article/details/124523779

Git的基础命令

- git clone xxx（xxx为刚刚复制的仓库链接）//代码下载到本地
- git branch xxx（xxx填写你的分支名称） //新建分支
- git branch -a //查看所有分支
- git checkout xxx（xxx填写要切换的分支名称）//切换到某一分支
- git add . //添加修改代码到缓存（注意最后的"."前面有个空格
- git commit -m "xxx" （xxx为本次提交代码的备注）//添加提交代码的备注
- git push origin xxx （xxx为要提交代码的分支名称）//提交代码到指定分支
- 。。。。

git clone

从远程仓库克隆一个版本库到本地

```
# 默认在当前目录下创建和版本库名相同的文件夹并下载版本到该文件夹下
$ git clone <远程仓库的网址>

# 指定本地仓库的目录
$ git clone <远程仓库的网址> <本地目录>

# -b 指定要克隆的分支，默认是master分支
$ git clone <远程仓库的网址> -b <分支名称> <本地目录>

```

git init

初始化项目所在目录，初始化后会在当前目录下出现一个名为 .git 的目录。

```
# 初始化本地仓库，在当前目录下生成 .git 文件夹
$ git init

```

git status

```
# 查看本地仓库的状态
$ git status

# 以简短模式查看本地仓库的状态
# 会显示两列，第一列是文件的状态，第二列是对应的文件
# 文件状态：A 新增，M 修改，D 删除，?? 未添加到Git中
$ git status -s

```

git remote

操作远程库

```
# 列出已经存在的远程仓库
$ git remote

# 列出远程仓库的详细信息，在别名后面列出URL地址
$ git remote -v
$ git remote --verbose

# 添加远程仓库
$ git remote add <远程仓库的别名> <远程仓库的URL地址>

# 修改远程仓库的别名
$ git remote rename <原远程仓库的别名> <新的别名>

# 删除指定名称的远程仓库
$ git remote remove <远程仓库的别名>

# 修改远程仓库的 URL 地址
$ git remote set-url <远程仓库的别名> <新的远程仓库URL地址>
```

git branch

操作 Git 的分支命令

```
# 列出本地的所有分支，当前所在分支以 "*" 标出
$ git branch

# 列出本地的所有分支并显示最后一次提交，当前所在分支以 "*" 标出
$ git branch -v

# 创建新分支，新的分支基于上一次提交建立
$ git branch <分支名>

# 修改分支名称
# 如果不指定原分支名称则为当前所在分支
$ git branch -m [<原分支名称>] <新的分支名称>
# 强制修改分支名称
$ git branch -M [<原分支名称>] <新的分支名称>

# 删除指定的本地分支
$ git branch -d <分支名称>

# 强制删除指定的本地分支
$ git branch -D <分支名称>
```

git checkout

检出命令，用于创建、切换分支等

```
# 切换到已存在的指定分支
$ git checkout <分支名称>

# 创建并切换到指定的分支，保留所有的提交记录
# 等同于 "git branch" 和 "git checkout" 两个命令合并
$ git checkout -b <分支名称>

# 创建并切换到指定的分支，删除所有的提交记录
$ git checkout --orphan <分支名称>

# 替换掉本地的改动，新增的文件和已经添加到暂存区的内容不受影响
$ git checkout <文件路径>
```

git add

把要提交的文件的信息添加到暂存区中；当使用 git commit 时，将依据暂存区中的内容来进行文件的提交。

```
# 把指定的文件添加到暂存区中
$ git add <文件路径>

# 添加所有修改、已删除的文件到暂存区中
$ git add -u [<文件路径>]
$ git add --update [<文件路径>]

# 添加所有修改、已删除、新增的文件到暂存区中，省略 <文件路径> 即为当前目录
$ git add -A [<文件路径>]
$ git add --all [<文件路径>]

# 查看所有修改、已删除但没有提交的文件，进入一个子命令系统
$ git add -i [<文件路径>]
$ git add --interactive [<文件路径>]
```

git commit

将暂存区中的文件提交到本地仓库中

```
# 把暂存区中的文件提交到本地仓库，调用文本编辑器输入该次提交的描述信息
$ git commit

# 把暂存区中的文件提交到本地仓库中并添加描述信息
$ git commit -m "<提交的描述信息>"

# 把所有修改、已删除的文件提交到本地仓库中
# 不包括未被版本库跟踪的文件，等同于先调用了 "git add -u"
$ git commit -a -m "<提交的描述信息>"

# 修改上次提交的描述信息
$ git commit --amend

```

git diff

比较版本之间的差异

```
# 比较当前文件和暂存区中文件的差异，显示没有暂存起来的更改
$ git diff

# 比较暂存区中的文件和上次提交时的差异
$ git diff --cached
$ git diff --staged

# 比较当前文件和上次提交时的差异
$ git diff HEAD

# 查看从指定的版本之后改动的内容
$ git diff <commit ID>

# 比较两个分支之间的差异
$ git diff <分支名称> <分支名称>
```

git pull

从远程仓库获取最新版本并合并到本地

git push

把本地仓库的提交推送到远程仓库

```
# 把本地仓库的分支推送到远程仓库的指定分支
$ git push <远程仓库的别名> <本地分支名>:<远程分支名>

# 删除指定的远程仓库的分支
$ git push <远程仓库的别名> :<远程分支名>
$ git push <远程仓库的别名> --delete <远程分支名>
```

git log

显示提交的记录

```
# 打印所有的提交记录
$ git log

# 打印从第一次提交到指定的提交的记录
$ git log <commit ID>

# 打印指定数量的最新提交的记录
$ git log -<指定的数量>
```

git mv

```
# 重命名指定的文件或者文件夹
$ git mv <源文件/文件夹> <目标文件/文件夹>
```

git rm

删除文件或者文件夹

```
# 移除跟踪指定的文件，并从本地仓库的文件夹中删除
$ git rm <文件路径>

# 移除跟踪指定的文件夹，并从本地仓库的文件夹中删除
$ git rm -r <文件夹路径>

# 移除跟踪指定的文件，在本地仓库的文件夹中保留该文件
$ git rm --cached
```

04 练习作业

实践练习

- 提交小组内每个人的 GitHub 账号；
- 提交个人基于开源创新思维实践课程项目提交的 Issue 链接地址；
- 提交个人点 Star 以及 Watch开源创新思维实践课程项目截图；
- 提交自己关注的开源项目地址。

感 谢 聆 听 ！

开 源 育 人 ， 共 创 数 字 梦 想 ！

扫码关注我们