Git使用

meng.fanding@qq.com 2019年3月8日

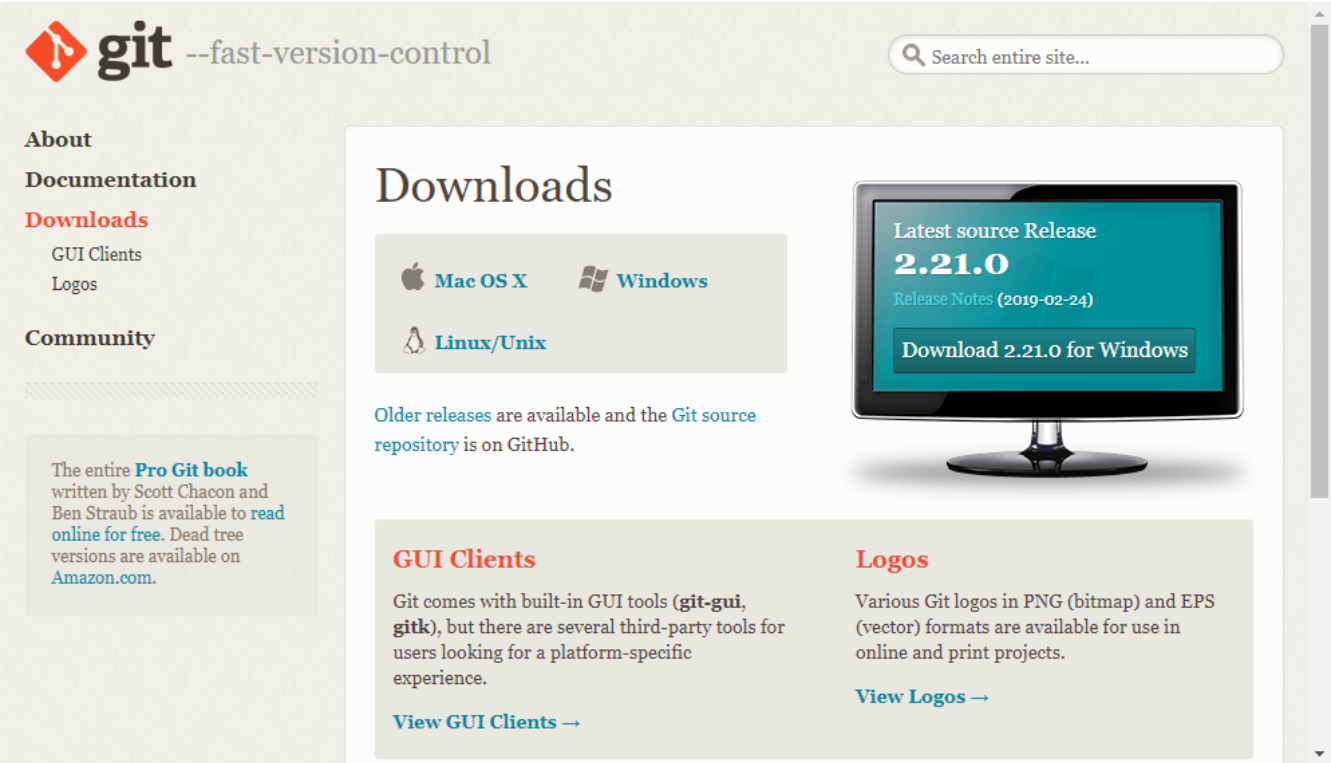
Git使用

- 一、开发
  1. 安装git.exe
  2. 生成SSH公钥
  4. 设置用户名，邮箱
  5. 设置GitLab密钥
  6. 提交代码到GitLab
  7. 常用操作
     1. 还原修改
     2. 部分commit后push失败
        1. 保留本地修改
        2. 放弃本地修改，直接覆盖之
- 二、测试
- 三、发布
  1. 分支发布
  2. 主干发布
- 四、使用场景
  1. 已经commit，返回上一版本
  2. 又不想还原了
  3. 恢复工作区修改的文件
  4. 已经add到暂存区需要还原
  5. 删除文件
  6. 误删工作区文件
  7. 分支

一、开发

1. 安装git.exe

git官网 下载对应版本，并安装



2. 生成SSH公钥

打开【Git bash here】

```
$ cd ~/.ssh
```

首次安装会出现找不到目录，跳过

```
bash: cd: /c/Users/aa/.ssh: No such file or directory
```

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/aa/.ssh/id_rsa):
Created directory '/c/Users/aa/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/aa/.ssh/id_rsa.
Your public key has been saved in /c/Users/aa/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:H8yZY7ZYjLXYP4YCLKXKuQqjvtvMYujmjAQN80f+EAA aa@DESKTOP-DEDBGLV
The key's randomart image is:
+---[RSA 2048]----+
|E..              |
|B=+.     . . .   |
|XO=+             |
+----[SHA256]-----+
```

过程中出现三次要求输入，直接回车

```
$ cat ~/.ssh/id_rsa.pub
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDIDWjWidNlPTldDQ0Fzwqfeo1wIEsxq4AlOhXDRnGynw34KDG2HMyc5sXO0yS9DeR7e
X1HS6R1ES4KNf1xDHQFuZTZtojP+Bbtm78ld8vvZwxtB0ARdf8fZ1E8Pt7Rl9XukyAOULDW2J13+u75TxR+CyMRVPR8d*****
********U1HUGKW+pdiTTWCwxyVOtgI64lwSf9fEhWoMX3SUIF6iOEtggISv0Fy7OKHjYSTPGPjfyTV2xxkJdsZr5Z89UNojB
1UqRoN/w8dxE0oG5TlSrc79J/KFuwPmuaV5JHDCebh8FPxyXHgx7 aa@DESKTOP-DEDBGLV
```

copy 以上 `ssh-rsa*****DEDBGLV` 部分

4. 设置用户名，邮箱

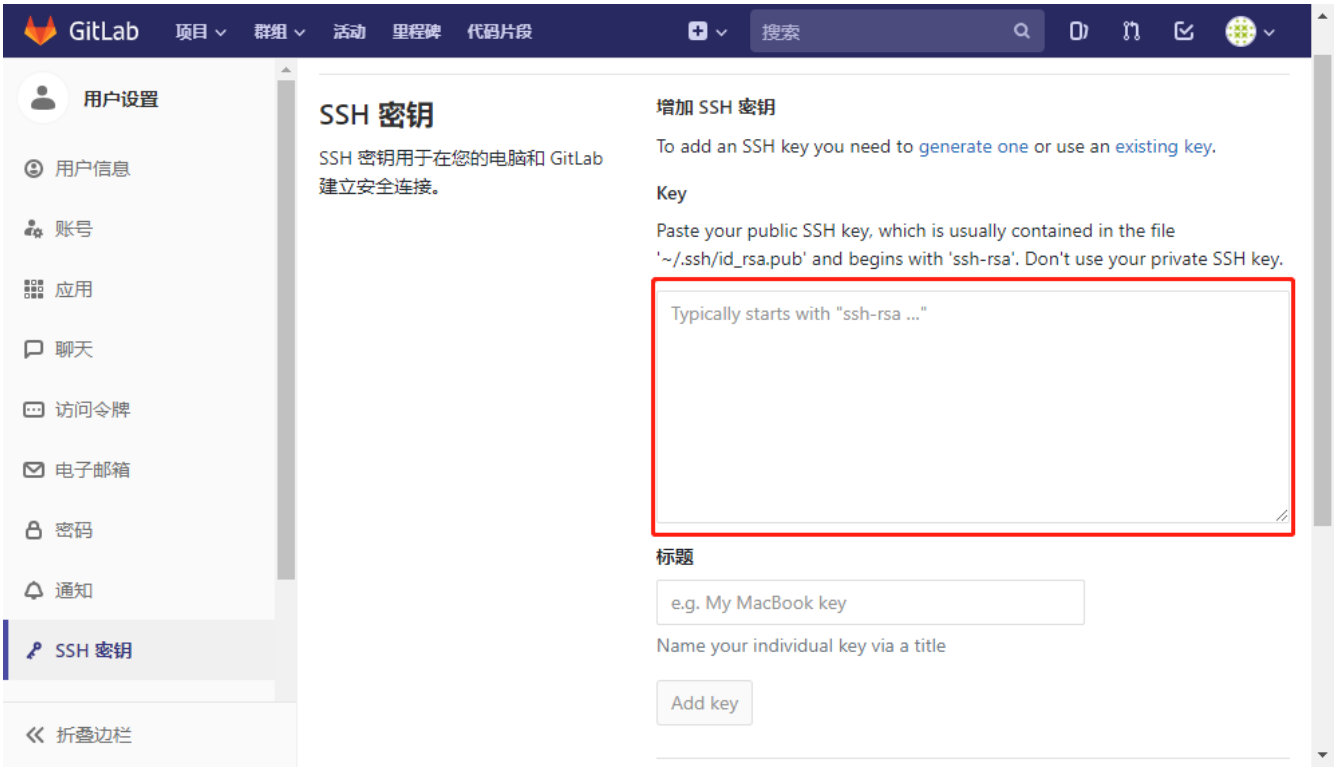
```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

5. 设置GitLab密钥

打开内部GitLab用户设置

http://192.168.0.242/profile

在【SSH密钥】中粘贴 `ssh-rsa*****DEDBGLV` 提交即可



6. 提交代码到GitLab

【略】

7. 常用操作

1. 还原修改

可引用git checkout或者用git clean -df至修改前的状态。就可以放弃所有修改。

1、 `git checkout` 功能是本地所有修改的。没有的提交的，都返回到原来的状态

2、 `git stash` 功能是把所有没有提交的修改暂存到stash里面。可用git stash pop回复。

3、 `git reset --hard HASH` 功能是返回到某个节点，不保留修改。

4、 `git reset --soft HASH` 功能是返回到某个节点。保留修改 。

5、 `git clean -df` 功能是保留修改，返回到某个节点。

2. 部分commit后push失败

1. 保留本地修改

```
git stash
git pull
git stash pop
```

接下来diff一下此文件看看自动合并的情况，并作出相应修改。

- git stash: 备份当前的工作区的内容，从最近的一次提交中读取相关内容，让工作区保证和上次提交的内容一致。同时，将当前的工作区内容保存到Git栈中。

- git stash pop: 从Git栈中读取最近一次保存的内容，恢复工作区的相关内容。由于可能存在多个Stash的内容，所以用栈来管理，pop会从最近的一个stash中读取内容并恢复。
- git stash list: 显示Git栈内的所有备份，可以利用这个列表来决定从那个地方恢复。
- git stash clear: 清空Git栈。此时使用gitg等图形化工具会发现，原来stash的哪些节点都消失了。

2. 放弃本地修改，直接覆盖之

```
git reset --hard
git pull
```

二、测试

开发提交到 `master` 分支，在master分支上打 `标签` 进行版本编译；创建上线版本分支；

三、发布

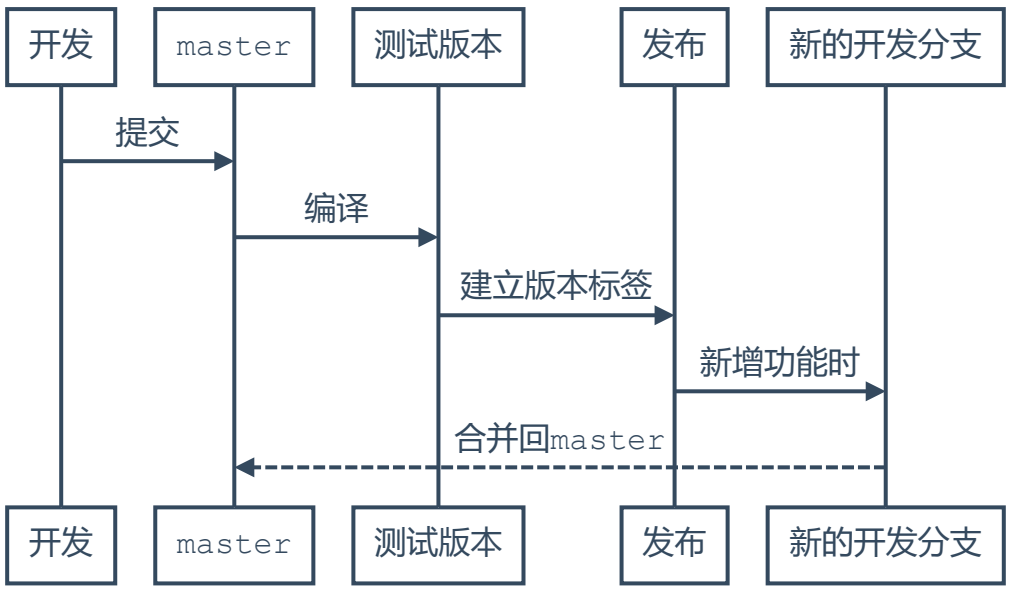
发布方式主要分两种，根据业务情况选择其一

1. 分支发布

得到一个稳定版本号后。将此稳定版本号放到一个新分支上，针对此稳定版本号的修修补补就在这个分支上进行。新功能不在此分支上开发。而在主干上进行新功能的开发。 这是业界采用较多的模式。

稳定分支上的有些改动。比方缺陷修复，须要合并到主干。 但有些特定改动，是不须要合并到主干的。这时须要千万注意，合并准确的文件到主干。

对于不能合并到主干的情况，常见的是再拉一个分支。这个分支专门为少数特定情况而用，但从全局讲，可能会导致太多分支。不同分支间混乱，所以这并不推荐。推荐宁愿采用配置开关。



2. 主干发布

得到一个稳定版本号后，拉出先锋分支，在分支上开发新功能。在主干上进行修修补补。

当先锋分支通过一定的测试之后，合并到主干。能够同一时候有多个先锋分支，不同的功能能够拉不同的分支，不同公布时间点而又要同一时候开发的内容必须在不同的分支上。

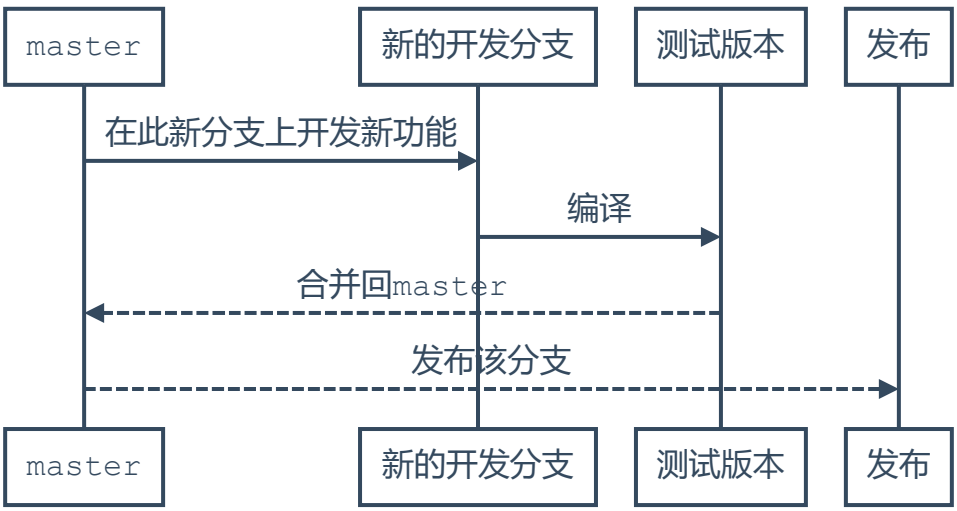
从公布的角度讲，更推荐将肯定一起公布的内容放在同样的先锋分支上。

主干上永远是稳定版本号，能够随时公布。bug的改动和新功能的添加，全部在分支上进行。并且每一个bug和新功能都有不同的开发分支。全然分离。

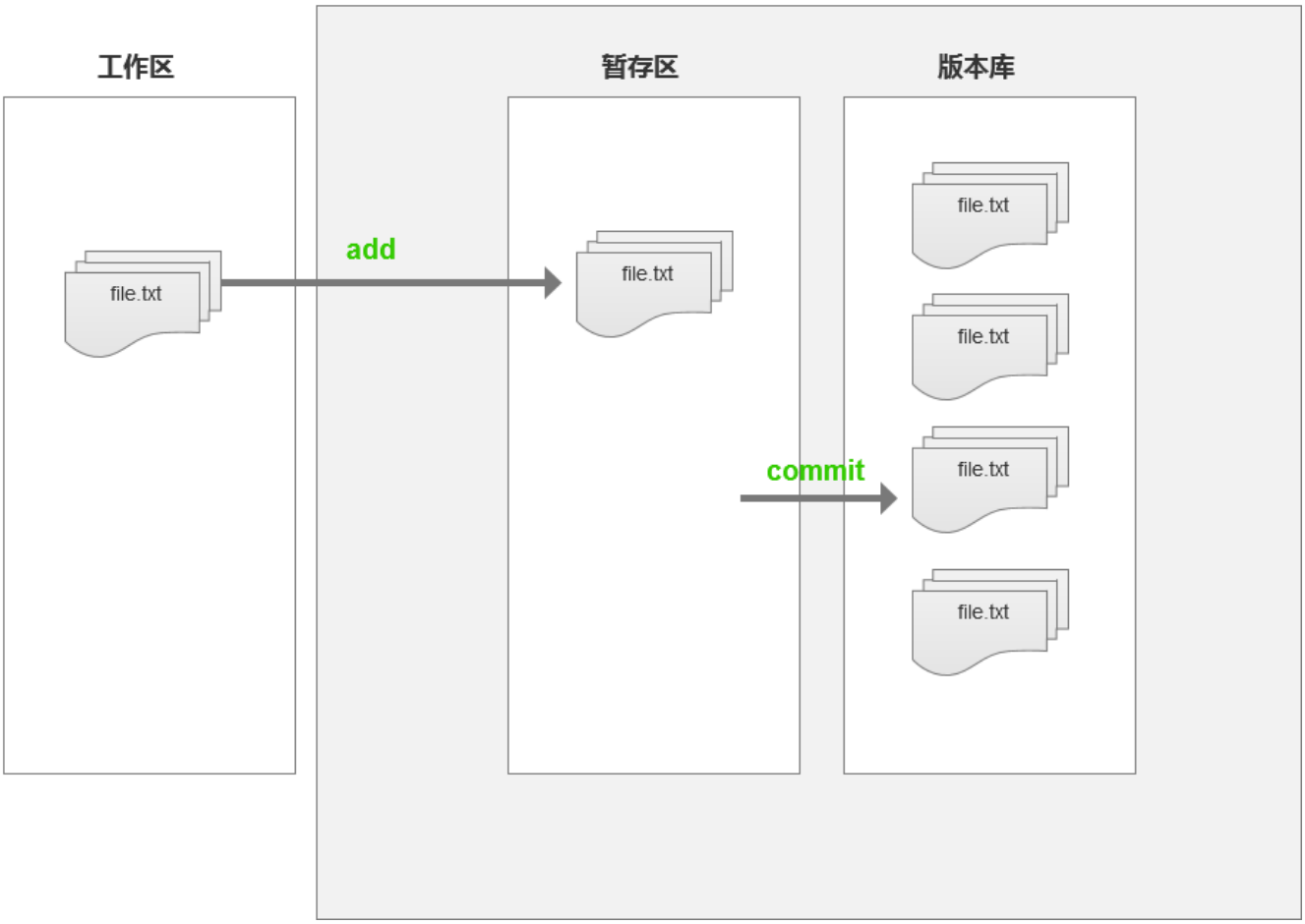
而对主干上的每一次公布都做一个标签而不是分支。分支上的开发和测试完毕以后才合并到主干。 这样的公布方法的优点是每次公布的内容调整起来比较方便。

假设某个新功能或者bug在下一次公布之前无法完毕，就不可能合并到主干。也就不会影响其它变更的公布。

另外，每一个分支的生命期比較短，唯一长期存在的就是主干。这样每次合并的风险非常小。每次公布之前，仅仅要比較主干上的最新版本号和上一次公布的版本号就能够知道这次公布的文件范围了。

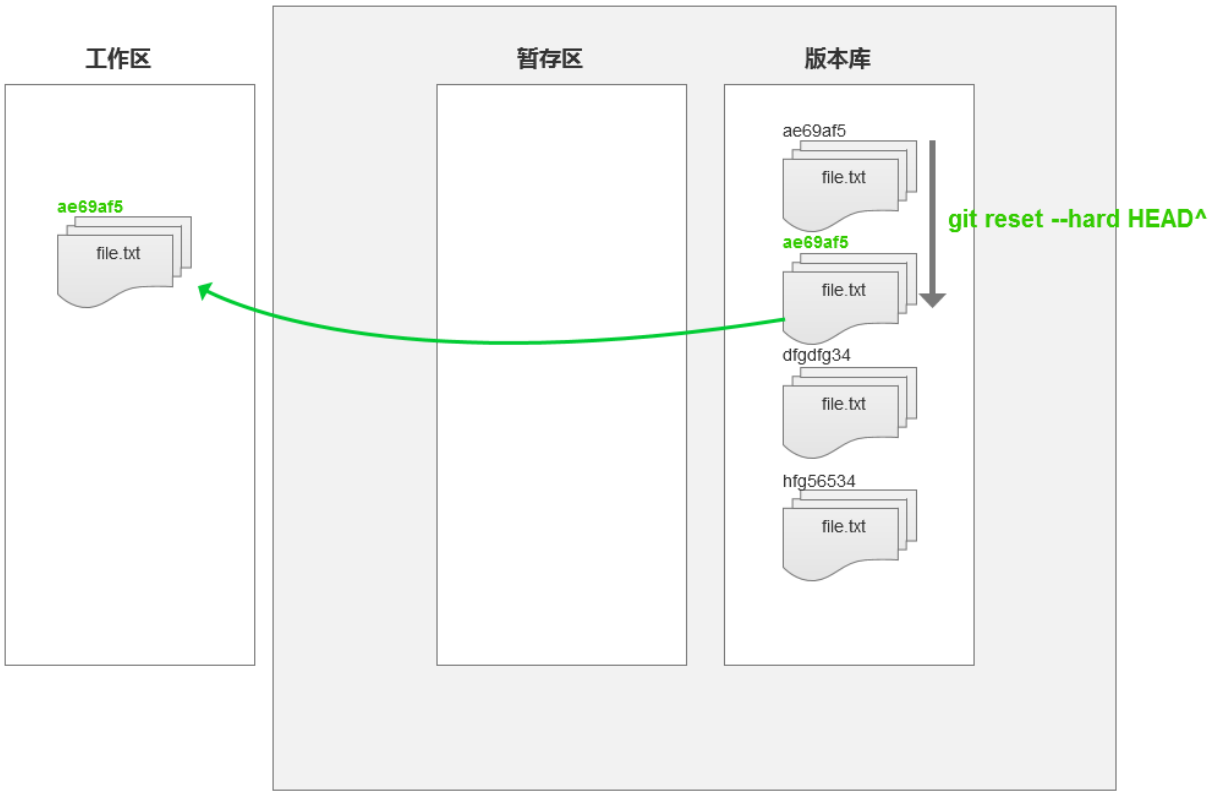


四、使用场景



1. 已经commit，返回上一版本

```
git reset --hard HEAD^
```



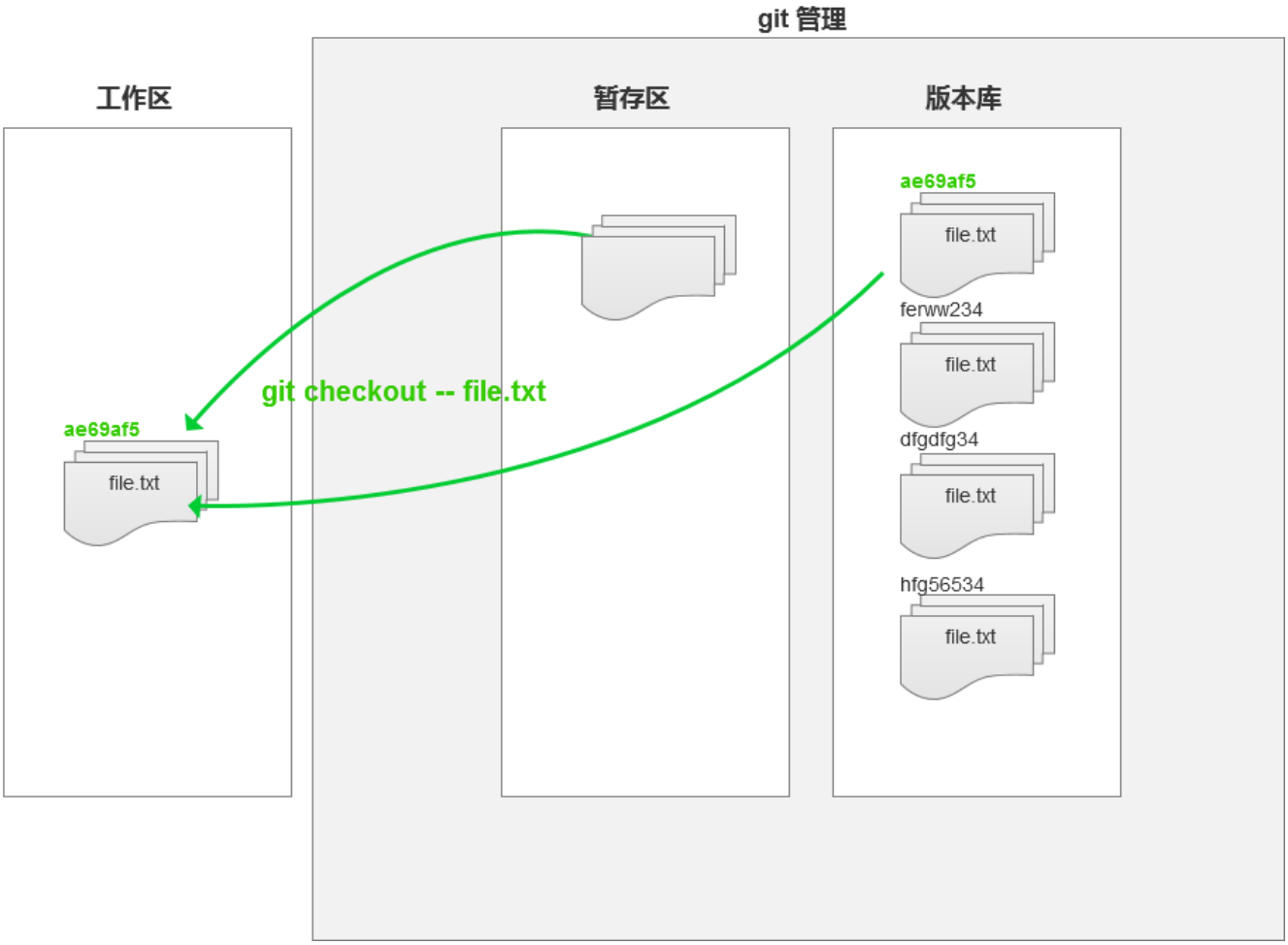
2. 又不想还原了

```
git reflog
git reset --hard [版本号]
```

3. 恢复工作区修改的文件

1. 还没有放到暂存区，回到和版本库一致状态
2. 已经add到暂存区，回到和暂存区一致状态

```
git checkout -- file.txt
```



4. 已经add到暂存区需要还原

1. 先把暂存区放回工作区

```
git reset HEAD test.txt
```

2. 再把工作区还原（见3）

5. 删除文件

1. 直接删除文件
2. add到暂存区
3. 提交

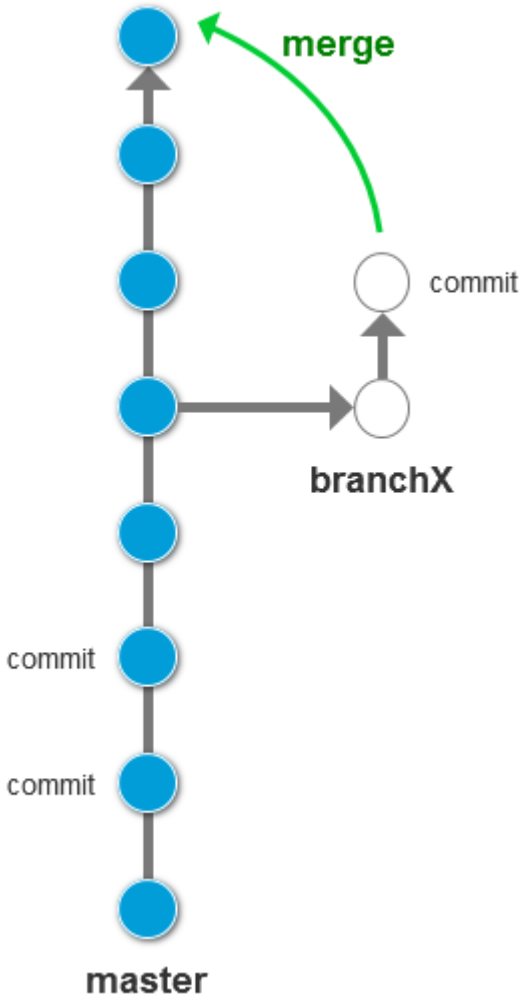
6. 误删工作区文件

```
git checkout -- file.txt
```

git rm 并且提交了之后，就需要使用git reset --hard 版本号 来回退

> `git checkout` 其实是用版本库里的版本替换工作区的版本

7. 分支



查看分支：`git branch`

创建分支：`git branch <name>`

切换分支：`git checkout <name>`

创建+切换分支：`git checkout -b <name>`

合并某分支到当前分支：`git merge <name>`

删除分支：`git branch -d <name>`