# Git读书笔记

### 一 设置Git的配置变量

<1> 设置用户名和邮件地址

**$ git config --global user.name “caogh1”**

**$ git config --global user.email “caogh1@lenovo.com”**

<2> 设置别名，以便使用更为简洁的子命令

# 别名被所有用户使用

**$ sudo git config --system alias.st status**

**$ sudo git config --system alias.ci commit**

# 只在本用户的全局配置中添加别名

**$ git config --global alias.st status**

**$ git config --global alias.ci commit**

**$ git config --global alias.co checkout**

**$ git config --global alias.br branch**

<3> 开启颜色显示

**$ git config --global color.ui true**

<4> 删除配置

**$ git config --unset --global user.name**

**$ git config --unset --global user.email**

Git的三个配置文件分别是版本库级的配置文件，全局配置文件（用户主目录下）和系统级配置文件（/etc目录下）。其中版本库级别的配置文件的优先级最高，全局配置文件次之，系统级配置文件优先级最低。Git config命令可以用于读取和更改INI配置文件的内容。使用只带一个参数的git config <section>.<key>命令可以来读取INI配置文件中某个配置的键值，更改某个属性的值，格式：git config <section>.<key> <value>。

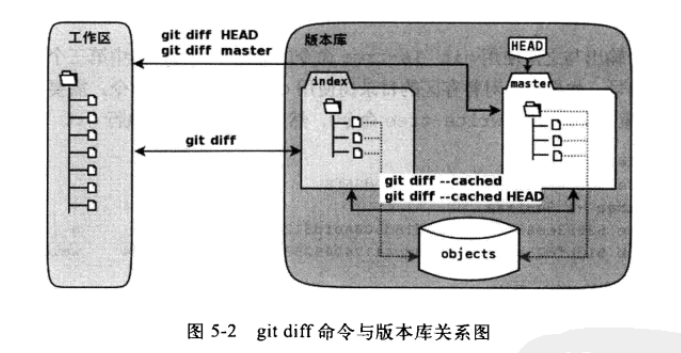
### 二 Git暂存区

文件.git/index记录了工作区文件的状态（实际上是暂存区的状态）。

M（Modified）标识的含义：

①位于第一列的字符M含义：版本库中的文件与处于中间状态---提交任务（提交暂存区，stage）中的文件相比有改动。

②位于第二列的字符M含义：工作区当前的文件与处于中间状态----提交任务（提交暂存区，stage）中的文件相比有改动。



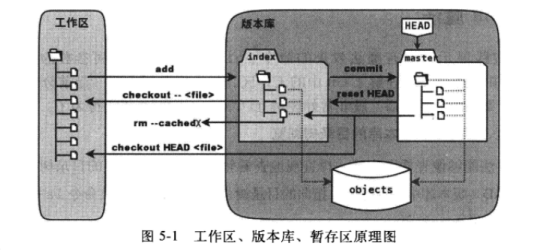
**通过不同的参数调用git diff命令可以查看不同状态下文件的差异**：

**①** 不带任何选项和参数调用git diff显示工作区的最新改动，即工作区与提交任务（提交暂存区，stage）中相比的差异。

**②** git diff HEAD将工作区和HEAD（当前工作分支）相比。

**③** 通过参数--cached或--staged调用git diff命令，查看的是提交暂存区（提交任务，stage）和版本库中文件的差异。

文件.git/index实际上就是一个包含文件索引的目录树，像是一个虚拟的工作区。在这个虚拟工作区的目录树中，记录了文件名和文件的状态信息（时间戳和文件长度等）。文件的内容并没有存储在其中，而是保存在Git对象库.git/objects目录中，文件索引建立了文件和对象库中对象实体之间的对应。



图中左侧为工作区，右侧为版本库。在版本库中标记为index的区域是暂存区，标记为master的是master分支所代表的目录树。HEAD实际是指向master分支的一个”游标“。Objects标识的区域为Git的对象库，位于.git/objects目录下。

■ 当对工作区修改（或新增）的文件执行git add命令时，暂存区的目录树将被更新，同时工作区修改（或新增）的文件内容会被写入到对象库中的一个新的对象中，而该对象的ID被记录在暂存区的文件索引中。

■ 当执行提交操作（git commit）时，暂存区的目录树会写到版本库（对象库）中，master分支会做相应的更新，即master最新指向的目录树就是提交时原暂存区的目录树。

■ 当执行git reset HEAD命令时，暂存区的目录树会被重写，会被master分支指向的目录树所替换，但是工作区不受影响。

■ 当执行git rm --cached <file>命令时，会直接从暂存区删除文件，工作区则不做出改变。

■ 当执行git checkout . 或git checkout -- <file>命令时，会用暂存区全部的文件或指定的文件替换工作区的文件，这个操作会清除工作区中未添加到暂存区的改动。

■ 当执行git checkout HEAD . 或git checkout HEAD <file>命令时，会用HEAD指向的master分支中的全部或部分文件替换暂存区和工作区的文件。这个命令不但会清除工作区中未提交的改动，也会清除暂存区中未提交的改动。

### 三 Git重置

Git reset命令可以改变分支引用文件的内容，即实现分支的重置。可以将”游标“指向任意一个存在的提交ID。

注：使用重置命令很危险，会彻底地丢弃历史。

Git提供了一个挽救机制，通过.git/logs目录下日志文件记录了分支的变更。默认非裸版本库（带有工作区）都提供分支日志功能。Git reflog 对.git/logs/refs/heads/master进行操作，可使用show子命令显示文件内容：

$ git reflog show master

Git reflog输出中最新改变放在最前面显示，而且只显示每次改变的最终的SHA1哈希值。表达式：<refname>@{<n>} 含义是引用<refname>之前第<n>次改变时的SHA1哈希值。

实例：

■ 重置master为两次改变之前的值

$ git reset --hard master@{2}

Gitreset命令的用法：

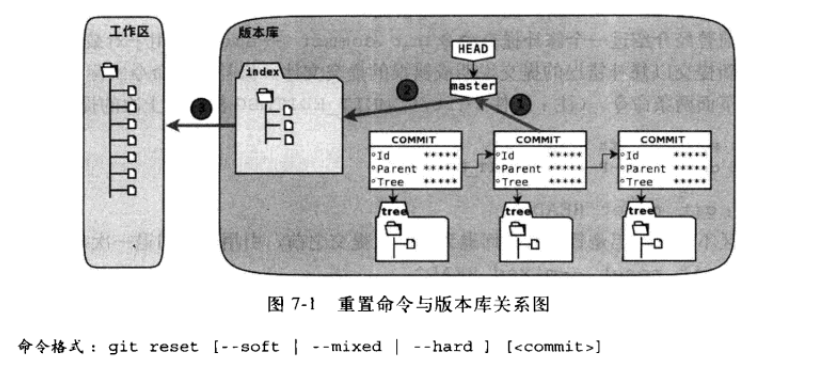
用法一：git reset [-q] [<commit>] [--] <paths> ...

用法二：git reset [--soft | --mixed | --hard | --merge | --keep] [-q] [<commit>]

其中<commit>都是可选项，可以使用引用或提交ID，如果省略<commit>则相当于使用了HEAD的指向作为提交ID。两种用法的区别在于，第一种用法在命令中包含路径<paths>。为了避免路径和引用（或者提交ID）同名而发生冲突，可以在<paths>前用两个连续的断线作为分割。

第一种用法（包含了路径<paths>的用法）不会重置引用，更不会改变工作区，而是用指定提交状态（<commit>）下的文件（<paths>）替换掉暂存区中的文件。例如命令git reset HEAD <paths>相当于取消之前执行的git add <paths>命令时改变的暂存区。

第二种用法（不使用路径<paths>的用法）则会重置引用。根据不同的选项，可以对暂存区或工作区进行重置。



■ 使用参数--hard，如：git reset --hard <commit>。

会执行上图中的全部动作①②③，即：

① 替换引用的指向。引用指向新的提交ID。

② 替换暂存区。替换后，暂存区的内容和引用指向的目录树一致。

③ 替换工作区。替换后，工作区的内容变得和暂存区一致，也和HEAD所指向的目录树内容相同。

■ 使用参数 --soft， 如：git reset --soft <commit>。

会执行上图中的操作①。即只更改引用的指向，不改变暂存区和工作区。

■ 使用参数 --mixed或不使用参数（默认为--mixed），如：git reset <commit>。

会执行上图中的操作①和操作②。即更改引用的指向及重置暂存区，但是不改变工作区。

**示例：**

◆ 命令：git reset 或 git reset HEAD

仅用HEAD指向的目录树重置暂存区，工作区不受影响，相当于将之前用git add命令更新到暂存区的内容撤出暂存区。引用也未改变，因为引用重置到HEAD相当于没有重置。

◆ 命令：git reset -- filename 或 git reset HEAD filename

仅将文件filename的改动撤出暂存区，暂存区中其他文件不改变。相当于对命令git add filename的反向操作。

◆ 命令：git reset --soft HEAD^

工作区和暂存区不改变，但是引用向前回退一次。当对最新提交的提交说明或提交的更改不满意时，撤销最新的提交以便重新提交。

◆ 命令：git reset HEAD^ 或 git reset --mixed HEAD^

工作区不改变，但是暂存区会回退到上一次提交之前，引用也会回退一次。

◆ 命令：git reset --hard HEAD^

彻底撤销最近的提交。引用回退到前一次，而且工作区和暂存区都会回退到上一次提交的状态。自上一次以来的提交全部丢失。

### 四 Git检出

重置命令reset的一个用途就是修改引用（如master）的游标指向。实际上在执行重置命令的时候没有使用任何参数对所要重置的分支名进行设置，这是因为重置命令实际上所针对的是头指针HEAD。之所以重置命令没有改变头指针HEAD的内容，是因为HEAD指向了一个引用refs/heads/master，所以重置命令体现为分支”游标“的变更，HEAD本身一直指向的是refs/heads/master，并没有在重置时改变。

检出命令git checkout实质就是修改HEAD本身的指向，该命令不会影响分支”游标“。

HEAD可以理解为”头指针“，是当前工作区的”基础版本“，当执行提交时，HEAD指向的提交将作为新提交的父提交。

“分离头指针”状态指的就是HEAD头指针指向了一个具体的提交ID，而不是一个引用（分支）。

检出命令git checkout会重写工作区，命令用法如下：

用法一：git checkout [-q] [<commit>] [--] <paths>...

用法二：git checkout [<branch>]

用法三：git checkout [-m] [[-b | --orphan]] <new\_branch> [<start\_point>]

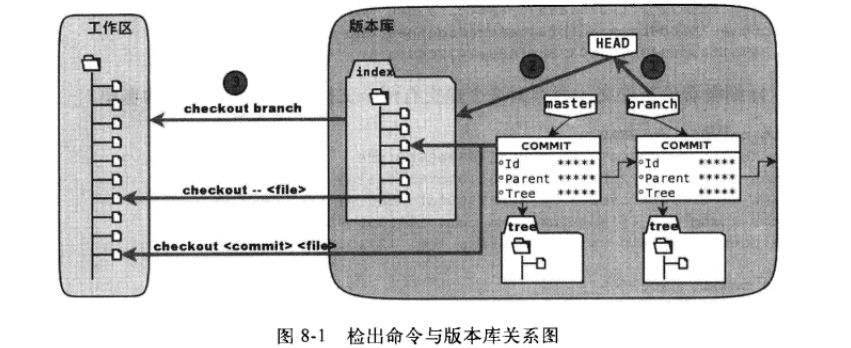
上面列出的第一种用法和第二种用法的区别在于，第一种用法在命令中包含路径<paths>。为了避免路径和引用同名而发生冲突，可以在<paths>前用两个连续的短线作为分割。

第一种用法的<commit>是可选项，如果省略则相当于从暂存区（index）进行检出。注：重置的默认值是HEAD，而检出的默认值是暂存区。因此重置一般用于重置暂存区（除非使用--hard参数，否则不重置工作区），而检出命令主要是覆盖工作区（如果<commit>不省略，也会替换暂存区中相应的文件）。

第一种用法（包含了路径<paths>的用法）不会改变HEAD头指针，主要是用于指针版本的文件覆盖工作区中对应的文件。如果省略<commit>，则会用暂存区的文件覆盖工作区的文件，否则用指针提交中的文件覆盖暂存区和工作区中对应的文件。

第二种用法（不使用路径<paths>的用法）则会改变HEAD头指针。之所以后面的参数写作<branch>，是因为只有HEAD切换到一个分支才可以对提交进行跟踪，否则仍然会进入“分离头指针”的状态。在“分离头指针”状态下的提交不能被引用关联到，从而可能丢失。所以用法二最主要的作用就是切换到分支。如果省略<branch>则相当于对工作区进行状态检查。

第三种用法主要是创建和切换到新的分支（<new\_branch>）,新的分支从<start\_point>指定的提交开始创建。新分支和master分支没有什么实质的不同。都是在refs/heads命名空间下的引用。



示例：

◆ 命令：git checkout branch

检出branch分支，完成上图的三个步骤，更新HEAD以指向branch分支，以及用branch指向的树更新暂存区和工作区。

◆ 命令：git checkout 或 git checkout HEAD

汇总显示工作区，暂存区与HEAD的差异。

◆ 命令：git checkout -- filename

用暂存区中filename文件来覆盖工作区中的filename文件。相当于取消自上次执行git add filename以来（如果执行过）的本地修改。

◆ 命令：git checkout branch -- filename

维持HEAD的指向不变，用branch所指向的提交中的filename替换暂存区和工作区中相应的文件。注意会将暂存区和工作区中的filename文件直接覆盖。

◆ 命令：git checkout -- . 或 git checkout .

注：git checkout命令后的参数为一个点。这条命令最危险，会取消所有本地的修改（相对于暂存区）。相当于用暂存区的所有文件直接覆盖本地文件，不给用户任何确认的机会！

定位工作区的根目录：

<1> 显示版本库.git目录所在

$ git rev-parse --git-dir

<2> 显示工作区根目录

$ git rev-parse --show-toplevel

<3> 相对于工作区根目录的相对目录

$ git rev-parse --show-prefix

<4> 实现从当前目录（cd）后退（up）到工作区的根的深度

$ git rev-parse --show-cdup

应用场景：

场景１：当已经存在一个git库，为了防止代码库误删，比较好的做法就是在本地建立一个备份，并经常与代码库同步。下列步骤将实现此功能。假设：已存在git代码库为[/home/caogh1/workspace/new\_hub]，

备份库位置：[/media/back/work/git\_workspace]

<1> 使用git init命令创建一个空的裸版本库。

git init --bare /media/back/work/git\_workspace/new\_hub.git

<2> 使用push操作为其创建内容。

cd /home/caogh1/workspace/new\_hub

git push --set-upstream /media/back/work/git\_workspace/new\_hub.git/ master

<3> 可以更新工作区内容，使用git push与备份库同步

场景2：当HEAD处于分离头指针时，处理如下。”分离头指针”状态指的就是HEAD头指针指向了一个具体的提交ID，而不是一个引用(分支)。

　　　git checkout master

如果在”分离头指针模式”下有提交，不能通过master分支或其他引用访问。如果提交是master分支所需要的，可以使用合并操作(git merge)，具体操作过程如下：

<1> 确认当前处于master分支。

$ git branch -v

<2> 执行合并操作，将提交合并到当前分支。

$ git merge <id>

<3> 仔细看看最新提交，会看到这个提交有两个父提交。这就是合并的奥秘。

$ git cat-file -p HEAD