# 基本概念

HEAD：指向你正在工作中的本地分支的指针

# git配置

1. 通过配置.gitignore文件中的内容，可以告诉git忽略某些文件（不进行跟踪）

|  |
| --- |
| $ cat .gitignore  \*.[oa]  \*~ |

文件 .gitignore 的格式规范如下：

* 所有空行或者以注释符号 ＃ 开头的行都会被 Git 忽略。
* 可以使用标准的 glob 模式匹配。
* 匹配模式最后跟反斜杠（/）说明要忽略的是目录。
* 要忽略指定模式以外的文件或目录，可以在模式前加上惊叹号（!）取反。

|  |
| --- |
| # 此为注释 – 将被 Git 忽略  # 忽略所有 .a 结尾的文件  \*.a  # 但 lib.a 除外  !lib.a  # 仅仅忽略项目根目录下的 TODO 文件，不包括 subdir/TODO  /TODO  # 忽略 build/ 目录下的所有文件  build/  # 会忽略 doc/notes.txt 但不包括 doc/server/arch.txt  doc/\*.txt  # 忽略 doc/ 目录下所有扩展名为 txt 的文件  doc/\*\*/\*.txt |

# git基本操作

## git branch

git branch 分支名称 创建对应的本地分支

git branch –[dD] 分支名称 删除分支。D为强制删除。 d如果有为合并的修改会删除失败

git branch –v 显示分支最后提交信息

git branch –a 查看所有分支，包括远程分支

git branch –r 查看远程分支列表

git branch –vv 查看本地分支对应的远程分支

## git checkout

用于切换分支或恢复代码到某一个提交。语法：git checkout [<tree-ish>] [--] <pathspec>…

git checkout –b <分支名> 创建分支并切换到该分支

git checkout -b newBranch origin/newBranch

git checkout <commit-id> 切换到某个commit-id所对应版本的代码

git checkout文件名 用于单独恢复某一个文件，--的作用就是告诉git这是一个文件

git checkout . 恢复所有文件

## git push

## 删除文件

使用git rm命令。会直接将文件从暂存区删除。然后可以通过git commit提交到本地版本库

如果希望删除的文件仅从暂存区中删除，而本地不删除，那么在git rm后增加—cached选项即可

## 移动文件

使用git mv命令，可实现文件重命名等操作

## 查看提交日志

使用 git log。-p选项可展开查看修改的内容 –n(n为具体数字。比如-2)，可查看指定条数的修改

--pretty 选项，可以指定使用完全不同于默认格式的方式展示提交历史。比如用 oneline 将每个提交放在一行显示，这在提交数很大时非常有用。另外还有 short，full 和 fuller 可以用，展示的信息或多或少有些不同，请自己动手实践一下看看效果如何。

按照时间作限制的选项，比如 --since 和 --until。下面的命令列出所有最近两周内的提交：

$ git log --since=2.weeks

查看git 提交日志可使用多种筛选方式。如下所示：

| **选项** | **说明** |
| --- | --- |
| -(n) | 仅显示最近的 n 条提交 |
| --since, --after | 仅显示指定时间之后的提交。 |
| --until, --before | 仅显示指定时间之前的提交。 |
| --author | 仅显示指定作者相关的提交。 |
| --committer | 仅显示指定提交者相关的提交。 |

## 撤销

场景1：

使用git commit的—amend选项，可以做实现如下场景的需求：

1. 忘记增加提交日志了，此时本地版本库、暂存区、工作区内容相同。使用git commit –amend可以补充增加日志
2. 有本地工作区的文件忘记提交了，此时先将本地工作区的文件通过git add加入到暂存区，然后使用git commit –amend选项可以提交文件。

上述操作在git log中只会产生一条提交记录。

场景2：撤销暂存区中的提交

有一些不想被提交的文件错误的提交到了暂存区，可以使用git reset HEAD <file>的方式撤销提交。不会改动本地工作区的文件内容

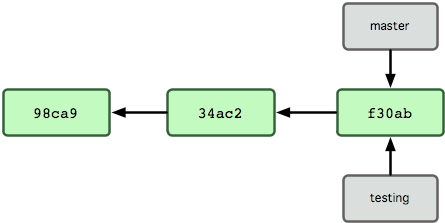
使用 git checkout <file> 会将本地工作区中的文件修改也撤销

## 分支管理

### 对git分支的理解

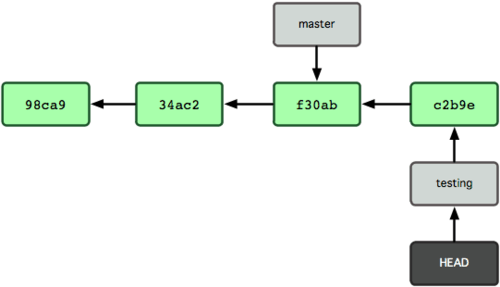
git分支不需要拷贝完整的代码（SVN需要复制完整的代码）

git的分支其实就像是一个指针。初始只有master分支，创建个分支后，分支相当于一个指针，指向master分支的某一个版本。比如在Master上创建test分支：

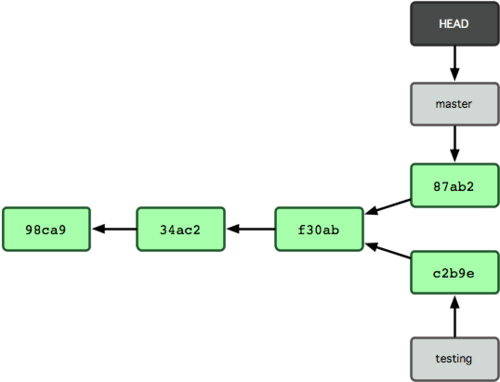


HEAD表明了目前工作在哪个分支。

在分支上提交修改，实际上就是移动了分支的指针



如果分支和主干都进行了不同的修改，树就会分叉



### 创建/删除/切换分支

使用git branch可以创建分支。使用 git branch –d选项可以删除分支

使用git checkout <分支名称> 可以将HEAD指向该分支，既认为当前在该分支下工作

创建并切换至分支，可以使用简化的命令 git checkout –b <分支名>

在git中，解决每个bug都可以通过在主干上创建对应bug分支的方法来屏蔽对master的影响。

### 分支合并

使用git merge命令来合并分支。在master中调用git merge <分支名称>来将分支中的修改合并到master。

一般分支与主干合并的步骤：

1. 在分支中合并主干的修改。
2. 切换的master，合并分支的修改。

### 解决冲突

当两个分支修改了相同位置的代码的时候，如果此时进行代码合并则会产生冲突

此时需要手动修改代码解决冲突，然后调用git add消除代码的冲突状态。

冲突状态的文件，通过git status查看状态是显示的是ummerge状态，简写为U

### 本地分支管理

git branch 不加任何参数可以查看所有分支。其中\*表示当前所在分支

使用git branch –merged查看已经合并的分支

使用git branch –no-merged查看未合并的分支。已合并的分支如果修改并提交后会变为未合并的分支。

### 远程分支

比如在Github上，创建仓库后默认会创建一个master分支。同时，还可以创建新的分支。比如创建分支Br1。那么master和Br1都属于仓库origin的远程分支。标签为：origin/master和origin/Br1

通过git remote可以查看关联的远程分支。git remote add可以添加远程分支到本地版本库。

比如

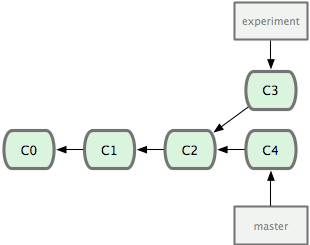
如果你在本地工作区创建了一个分支testBr。你希望将该分支添加到远程仓库中成为远程分支，可以使用git push <远程仓库名> <远程分支名>

git push不加参数时，会将代码推送到git remote的所有远程仓库

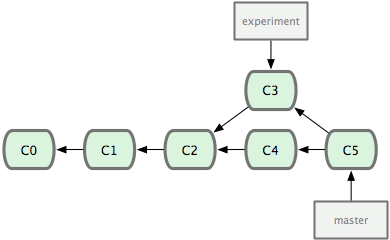
git branch 可以查看本地分支， git branch -r  可以用来查看远程分支

### 分支的变基

合并分支代码的方式可以通过merge，也可以通过rebase。rebase成为变基。实际上就是改变分支的基准。

 分支的修改

merge后



rebase后



变基的原理的是回到两个分支最近的共同祖先，根据当前分支（也就是要进行变基的分支 experiment）后续的历次提交对象（这里只有一个 C3），生成一系列文件补丁，然后以基底分支（也就是主干分支 master）最后一个提交对象（C4）为新的出发点，逐个应用之前准备好的补丁文件，最后会生成一个新的合并提交对象（C3'），从而改写 experiment 的提交历史，使它成为 master 分支的直接下游

rebase于merge相比，提交记录更加简洁。就好像在一条线上开发一样。

### Cherry Pick

应用背景：假设现在有两个分支：dev\_01, dev\_02. 如果我想把dev\_01分支上的某几个commit合并到dev\_02分支， 那么怎么办呢? cherry-pick会捡选某些commit, 即把某些commit的更改重新在另一分支上重新提交一遍. 对于我们的应用背景而言，就是把分支dev\_01上某几个commit的更改在分支dev\_02上再提交一遍

cherry-pick命令每拣选一个commit就会提交一次生成一个新的commit id。 如果我们想让每个commit 拣选后暂缓提交，等到所有commit都拣选完成后，自己手动commit，应该怎么办呢？答案是用-n 选项：

## 远程库管理

### git clone

使用git clone克隆远程仓库，远程仓库克隆后的默认名称为origin。可以通过-o选项修改。

### git remote

可以通过git remote查看所有远程仓库

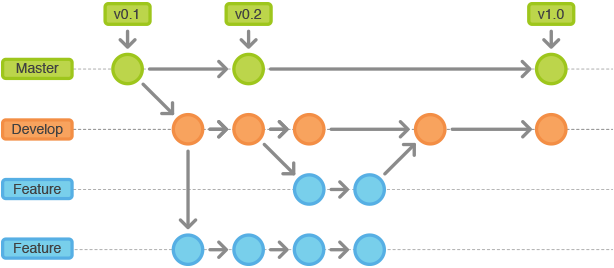
# GitFlow工作流程

两个分支：Master分支和Develop分支。

Master分支永远和线上版本保持一致

Develop分支合并相关功能修改。在开发新的功能的时候，应该在develop分支上派生自己的开发分支。而不应该在Master上创建分支

功能开发永远不应该直接牵扯到master



# QA

1. git remote查看的是远程的什么？

git remote查看远程仓库。

你克隆的分支默认为origin。无论这个分支在远程仓库中是主干分支还是其他分支，克隆到本地后，该分支就是本地的origin

可以通过git remote add添加远程仓库。添加远程仓库的意义在于你可以向不同的远程仓库推送代码修改。

1. git checkout 和git clone的区别
2. git fetch和git pull的区别

FETCH\_HEAD： 是一个版本链接，记录在本地的一个文件中，指向着目前已经从远程仓库取下来的分支的末端版本。

commit-id：在每次本地工作完成后，都会做一个git commit 操作来保存当前工作到本地的repo， 此时会产生一个commit-id，这是一个能唯一标识一个版本的序列号。 在使用git push后，这个序列号还会同步到远程仓库。

有了以上的概念再来说说git fetch

git fetch：这将更新git remote 中所有的远程仓库所包含分支的最新commit-id, 将其记录到.git/FETCH\_HEAD文件中

使用git fetch拉取远程仓库中的内容的步骤如下：

|  |
| --- |
| git fetch origin master:tmp  //在本地新建一个temp分支，并将远程origin仓库的master分支代码下载到本地temp分支  git diff tmp  //来比较本地代码与刚刚从远程下载下来的代码的区别  git merge tmp  //合并temp分支到本地的master分支  git branch -d temp  //如果不想保留temp分支 可以用这步删除 |

而git pull则是上述步骤的合并。git pull 用法

|  |
| --- |
| git pull <远程主机名> <远程分支名>:<本地分支名> |

git fetch用法：

|  |
| --- |
| 1. git fetch这将更新git remote 中所有的远程repo 所包含分支的最新commit-id, 将其记录到.git/FETCH\_HEAD文件中  2. git fetch remote\_repo  这将更新名称为remote\_repo 的远程repo上的所有branch的最新commit-id，将其记录。  3. git fetch remote\_repo remote\_branch\_name这将这将更新名称为remote\_repo 的远程repo上的分支： remote\_branch\_name  4. git fetch remote\_repo remote\_branch\_name:local\_branch\_name这将这将更新名称为remote\_repo 的远程repo上的分支： remote\_branch\_name ，并在本地创建local\_branch\_name 本地分支保存远端分支的所有数据。 |