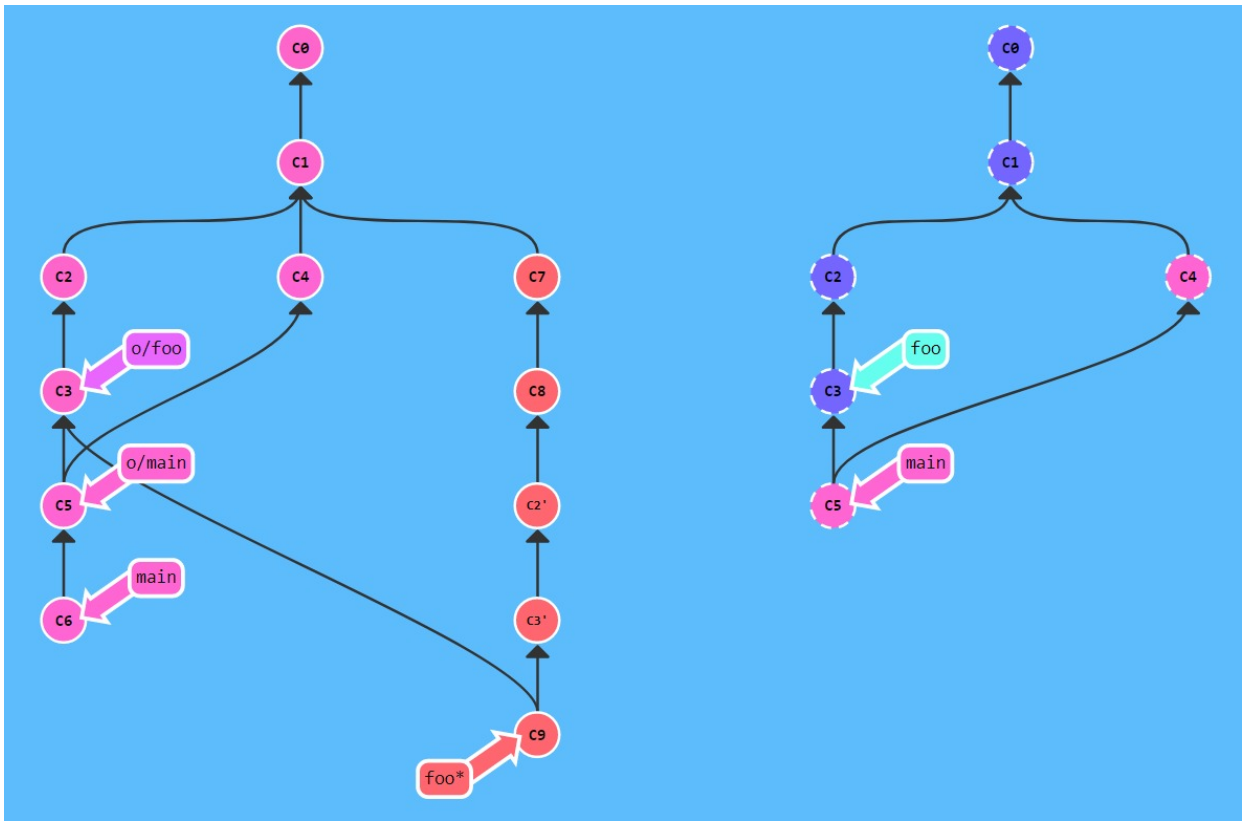
1 基本命令与技巧



git commit 递交更改，向前推进一个节点（提交记录）

git branch 查看所有分支
git branch *newbranch* 在检出处新建一分支
git branch -f *movebranch refer* 将分支强制指向另一个节点（*refer*默认为HEAD所指）

git checkout *refer* / git switch *refer* 切换（检出）到另一个节点或分支
refer : 用名称（分支、标签、哈希值）或相对位置标识的某个节点
git checkout -b *newbranch refer* 在*refer*（默认为检出处）新建一分支并切换至该新分支（检出一新分支）

---

git merge *frombranch (fromrefer)* 合并两个分支，由所在处创建一个并入提交 或 当*frombranch*为子节点（继承自所在处）时直接更新至末端（快进 Fast-forward 策略）即可

git rebase *ontobranch (ontorefer)* 将一个分支的更改接续（实为新的提交）到另一个分支上 或 当*ontobranch*为子节点时直接快进即可
git rebase *ontobranch movebranch (ontorefer moverefer)* 可以无须检出，指定要移动接续的分支（执行完后HEAD指在*movebranch*）

HEAD 总是指向当前分支最近的一次提交记录
当执行检出（checkout / switch）至某处的命令时可能会出现HEAD分离
一般正常情况下有 HEAD -> *branch* -> *Hash (commitID)*

相对引用 ***refer +*** :
^ 向上移动一个节点 **~*num*** 向上移动若干个节点（默认*num*为1） **^*num*** 移动至其一级父节点中的第*num*个
可以连写表示一条较长的完整路径

git reset *refer* 让当前分支回退至某节点，可理解为重置至某处（HEAD不能分离*branch*）

git revert *refer* 撤销（提交抵偿操作）指定节点记录，共享的远程分支应采用此方式

---

git cherry-pick *refer(s)* 在当前检出（HEAD）处添加除HEAD上游以外任意节点的更改，可自选整理提交树
（*refers* : 等价于 *refer1 refer2 ...*）

git rebase -i *ontorefer* 在*ontorefer*处新接建子树并将当前*branch*（或分离的HEAD）指向其末端，调出交互式界面整理提交树（可以调整、删除、合并提交节点，范围从当前检出处至*ontorefer*或其共同父节点）

Squash 压缩提交的方法:

1. 交互式变基来重整 **git rebase -i**
2. **git merge --squash *branch*** 不会自动创建一个合并提交，而把源分支所有修改放入暂存区（再进行提交，相当于压缩为一个节点）

常见修正上游的技巧 :

- **git rebase -i** 调换顺序至末端；**git commit --amend** 追加提交，为同级子节点（不想要原提交时）；**git rebase -i** 再调换回原顺序
- 在特定节点 **git commit --amend** 完后 **git cherry-pick** 追回原本后来的提交

---

git tag *version refer* 给某个特定的提交创建标签（*refer*默认为HEAD所指），类似提交树上的锚点，不会改动且HEAD不会指向标签（用标签作*refer*便会出现HEAD分离）

git describe *refer* 给出某个特定节点上游最近的标签（*refer*默认为HEAD所指），输出格式为 *tag_distance_gHash*（*distance* : 该节点与标签相差的提交数 *Hash* : 该节点的哈希值）

2 远程仓库相关

git clone *url* 在本地创建一个远程仓库的拷贝并产生远程分支
远程分支：（本地）反映了远程仓库（在上一次通信时）的状态（同样由于不可操作，检出会自动进入HEAD分离），命名格式为 remote/branch（*remote*默认为 **origin**），*对应remote仓库里的branch分支*

git fetch 下载远端所有的提交记录更新到本地仓库中的远程分支，尚未修改本地文件

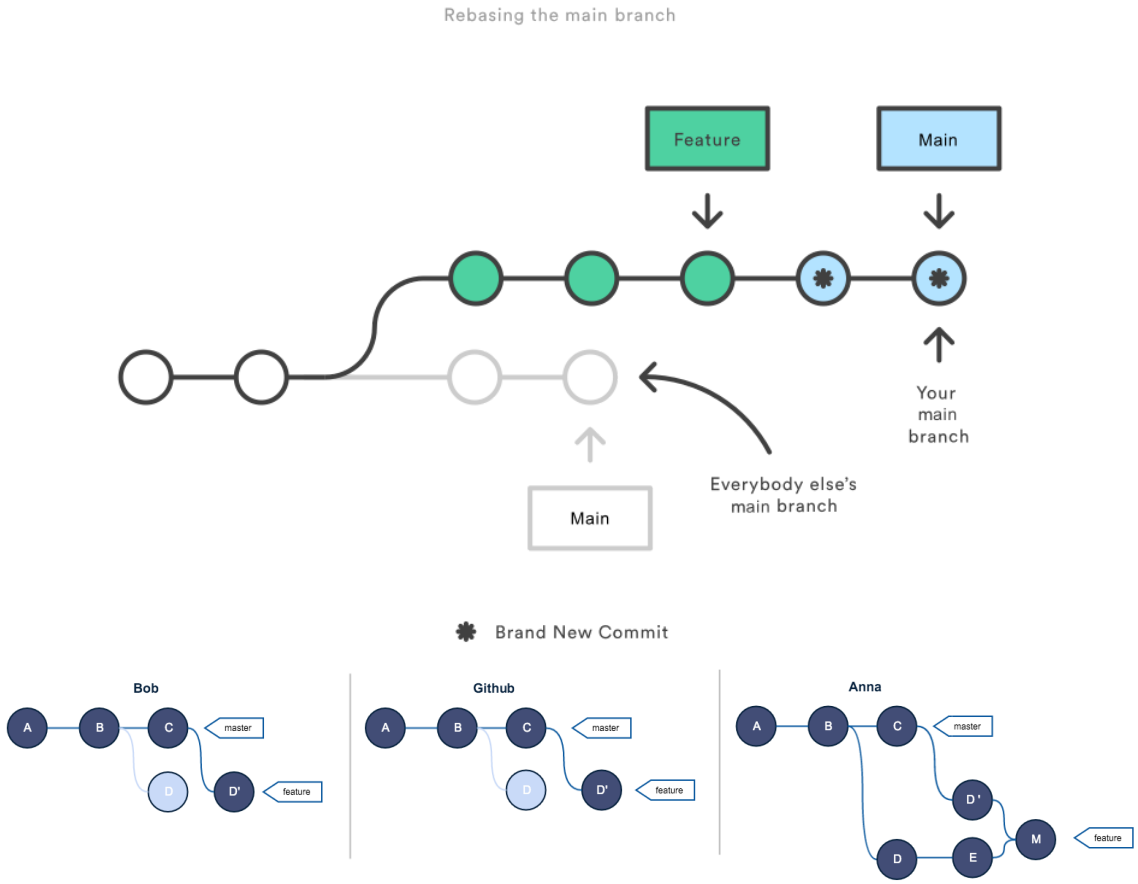
git pull 等同于 fetch + merge
git pull --rebase 等同于 fetch + rebase

git push 将本地变更记录上传到远端，同时更新远程分支
当远端已有新的变更造成冲突时，需要先 pull 下来使本地工作基于最新的远程分支，再进行 push 上传分享
常见工作流 : 在多个特性分支*feature*上工作，完成后进行一次集成 或 先合至*main*，只在*main*上推拉，保持最新

很多时候远端项目*main*主支锁定（无权限），需要推送到一个新建分支，并申请 PR（pull request）来更新
! [remote rejected] main -> main (pre-receive hook declined)

merge 与 rebase :

- 皆用于引入上游分支（合作者）的修改
- rebase 保持了提交树的线性，而 merge 在多分支下变得较为复杂
- merge 方便追溯起因，而 rebase 重写了分支的历史，修改了提交的顺序
- *rebase黄金法则* : 永远不要在一条共享的分支上使用它



（示例中*Bob*需要使用 **git push --force** 强行摘掉原分支以解决*feature*上的冲突）

- 已 PR 的分支将受到审查，类似公共分支而不应再 rebase，但可以通过一次交互式 rebase 清理提交历史
  审查通过将要并入时，可以选择先 rebase 到*main*再 merge 快进，从而保持线性历史（或直接 merge）

远程跟踪 : 本地分支关联于一个远程分支，这指定了推送及拉取的目标
local branch "main" set to track remote branch "origin/main"
设定跟踪属性的方法 :

- **git checkout -b *newbranch remote/branch*** 用远程分支检出新分支，其将跟踪该远程分支
- **git branch -u *remote/branch trackbranch*** 设定*trackbranch*的跟踪对象（*trackbranch*默认为HEAD所指分支（需要为分支））

---

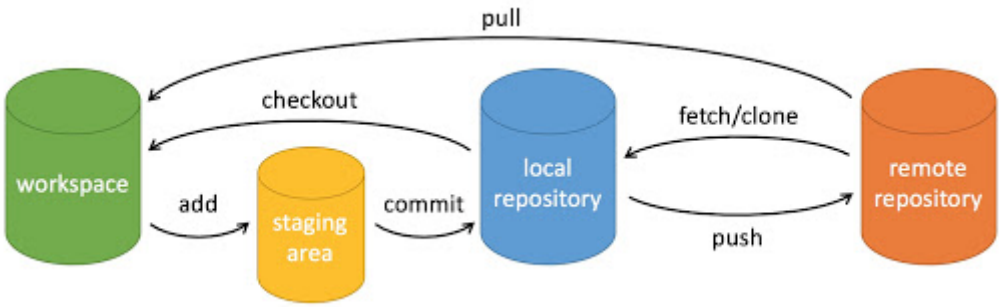
git push *remote mybranch* 指定推送*mybranch*到*remote(origin)*仓库的对应跟踪的远支
git push *remote myrefer:ourbranch* 从*myrefer*处的记录推送到*remote*上*ourbranch*，当分支名不存在时会新建

git fetch *remote ourbranch* 从*remote*下载指定分支*(origin/)ourbranch*
git fetch *remote ourrefer:mybranch* 从*remote*指定位置*ourrefer*下载并更新到指定分支*mybranch*（如果是当前检出分支会HEAD分离），当分支名不存在时会新建

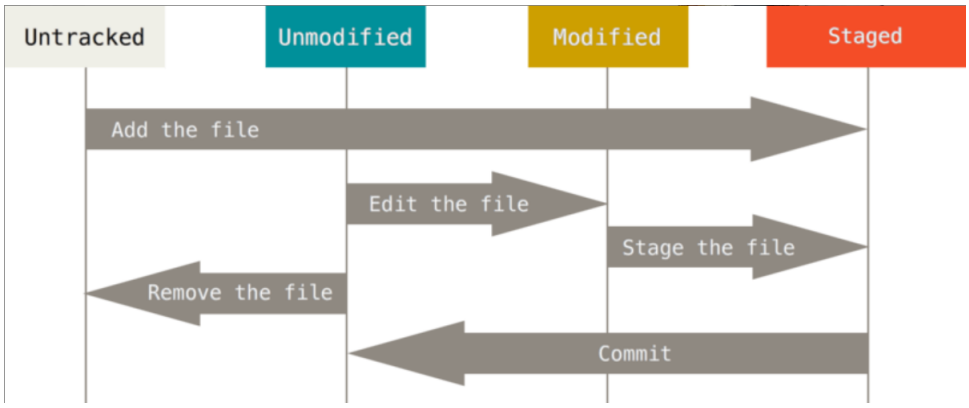
git push *remote* :*delbranch* 删去远端的*delbranch*及本地对应远程分支（用冒号明确映射后HEAD不会改动）
git fetch *remote* :*newbranch* 在本地检出处新建*newbranch*

git pull *remote branch|map* 等同于 fetch + merge（merge本地更新的分支）

3 工作管理与Github



remote (repo) 远程仓库，可视作位于网络的另一台主机上（*GitHub*）
local repo 本地仓库，或称版本库，记录有提交历史和分支，由 .git 文件夹（隐藏）管理
stage / index 暂存区，或称索引，可以暂存修改后预提交的文件，实际只是一个索引文件（.git/index），包含文件目录树的快照，指向写有文件修改内容的对象库（.git/objects）
worktree / workspace 工作空间，即查看的仓库文件夹与进行编辑修改的地方



文件有四种状态，通过 **git status** 可以查看目前各区文件情况：
untracked 未跟踪，如新加入文件夹，但未参与版本控制，**git add** 后加入暂存区为 staged
unmodified 入库状态，无修改，若修改后变为 modified，**git rm** 将移出版本库变为 untracked
modified 修改，可以 **git add** 加入暂存，**git commit -am** 直接递交，也可以 **git checkout** 取出库文件覆盖（丢弃当前修改）
staged 暂存状态，**git commit** 将入库，此时本地均一致，变为 unmodified，也可以 **git reset HEAD *file*** 取消暂存（退回 modified）

Bash 基础：

ls 列出当下目录中的文件(夹)，**ls -a** 列出全部（包括隐藏）的文件(夹)，**ls -l** 列出文件(夹)及其相关详细信息

cat *file(s)* 显示若干文本文件的内容（*file*可以包括*path*），**cat *file(s)* -n** 能标上行号显示
cat *file(s)* > *newfile* 将若干文件内容重定向到一文件中，这会覆盖或新建，使用 **>>** 可以追加

head *file* **tail *file*** 可以显示文件的首或尾十行

diff *filea fileb* 比较两个文件的差异

mv *oldfile newfile* 移动文件，也可重命名
cp *srcfile copyfile* 复制文件，也可重命名
rm *file* 删除文件，在目录上需使用 **-rf** 递归删除

gzip|gunzip *file* 压缩或解压文件
gzcat *file* 不解压情况下查看文件

mkdir *dir* 生成一个新目录
cd *dir* 切换到指定目录，**cd** 默认切换至主目录，可用 **.** 表示当前目录，**..** 表示上级目录
pwd 查看当前工作目录

clear 清屏

文本操作、系统与网络、管道、Shell脚本、Vim等可自行搜索参考

建仓：

git init 当前所在目录建仓（自动为*main*分支）

git clone *url projectname* 拷贝一个 Git 仓库到本地，在当前目录下生成该项目目录，可自己另命名

修改与提交：

git add *file(s)|dir* 添加若干个文件或指定目录（及其子目录）至stage
git add . 当前目录下所有文件至stage（也可以使用 ***** ）

git status 与上次提交相比，worktree和stage文件的更改情况
git status -s 获得简短输出结果（**A** 已加入缓存，**M** 有改动未加入缓存，**AM** 加入缓存后又有改动）

git diff *file(s)* 比较指定文件的差异，无任何参数时显示工作区与暂存区所有文件内容的差异（或称尚未缓存的全部改动），*file(s)* 也可写作 *-- file(s)*（--后有空格作分隔）
git diff --cached 查看已缓存的改动
git diff HEAD 查看已缓存和未缓存的所有改动（直接去比较版本库）
git diff *firstrefer secondrefer file(s)* 比较本地仓库中两处的差异，*secondrefer*没有时则与工作空间比较，*file*默认为全体文件
git diff --stat 显示摘要（文件列表，和其正负修改的数量）
（注意命令的可选参数可以组合使用，下同）

git commit *file(s)* -m *message* 将暂存区文件（*file(s)*默认为全体）添入本地仓库，形成新节点，且备注有提交信息
git commit -am *message* 可以跳过 add 把工作区直接提交到版本库，无备注信息时只需 **git commit -a**
git commit --amend 对最近一次提交的信息与修改内容进行更正，而不添加新节点（更换了 commitID）

git reset --soft|mixed|hard *refer* HEAD与分支回退到指定版本（*refer*默认原地），可选项默认为mixed
soft 仅回退仓库，即回退差异放入暂存区，回退后再commit可以将多个提交整合为一个
mixed 回退仓库与暂存区，回退差异放入工作空间便于即刻更改，常用 **git reset [HEAD] *file(s)*** 来移除（**unstage**）所有暂存区待提交文件
hard 回退本地包括暂存区和工作区的所有改变，强制放弃目标节点后的更改，恢复当时工作状态（可以 **git reflog** 显示所有提交节点，查询到现漂流的commitID重设回去）

git rm *file(s)* 从工作区和暂存区中将文件删除，仍待提交至仓库（如果手工从工作目录里删去文件，会视作 *changes not staged for commit*）
git rm -f *file(s)* 若本地有 modified，需要 -f 强制才能从两区删除
git rm --cached *file(s)* 仅从暂存区删除而保留工作区

git mv *file newfile* 移动或重命名一个文件或目录（*file*可以包括*path*）
git mv -f *file newfile* 当新文件名已存在时，可以 -f 强制覆盖

回溯：

git log 查看历史提交日志，常见参数选项有：

- **--oneline** 简洁版本（哈希值前几位和提交信息）
- **--graph** 绘制一个ASCII树形结构展示提交历史
- **--stat** 输出文件正负修改数据
- **--decorate** 输出提交的分支和标签
- **-p** 以 diff 形式详细输出提交的修改内容
  git show *refer* 等价于 log -p，但仅显示一个提交的内容（*refer*默认为HEAD所指）
- **--author --before --after** 按作者或时间段过滤提交记录
- **--no-merge** 略去合并提交记录
- **--grep *regex*** 以正则表达式过滤提交记录

git blame 逐行列出其修改的来源（作者、时间、提交）

git tag 查看所有标签
git tag -a *version* -m *message* 给当前提交打上带注解（作者、时间、信息）的标签，略去 -m 将打开编辑器（加*refer*可以在该处追加标签）
git tag -d *version* 删除指定标签

分支与管理（补充 **1**）：

git branch -d *delbranch* 删除版本库中的分支，若此分支没有 merge 过改动将丢失，-d 会提示失败（**-D** 等同于 **-d -f**，能够强制删去）

git checkout 有两大功能：

1. 分支检出
   git checkout *refer* 检出至仓库某特定节点，覆盖暂存区和工作区，当两区内有未提交的代码时，检出别处会被阻止，**-f** 能强制覆盖
2. 文件恢复
   git checkout *file(s)* 从暂存区恢复文件到工作区，**git checkout .** 恢复全部（放弃工作区所有更改）
   git checkout HEAD *file(s)* 从仓库中恢复文件到暂存区和工作区

新版 Git 拆分出了两个新命令：

git switch *branch* 切换到某分支（无法切换到ID，除非使用 **--detach**）
git switch -c *newbranch refer* 等同于 checkout -b
git switch - 切换至前一个切换的分支

git restore [--worktree] *file(s)* 从暂存区恢复文件状态到工作区
git restore --staged *file(s)* 从仓库恢复文件状态到暂存区（再加上 **--worktree** 可同时恢复两区）

合并、变基等分支操作要求暂存区不能留有待提交内容，否则会失败（可先贮存）

git merge *branch* -s *strategy* -m *message* 选择合并的策略，没有 -m 会打开编辑器编写 merge commit 信息

合并策略：

- **recursive** 默认策略，递归三向合并，参考共同父节点判断在哪些地方作了更改，如果共同父节点不唯一会继续递归这些节点的共同父节点直至唯一，再逐步（以前一层结果为参考点）合并回来，当出现冲突时暂停（**ort** 策略为其优化重构）（**subtree**
- **resolve** 多个共同父节点时会选择其中一个
- **ours** 冲突直接听取当前分支作为结果
- **octopus** 允许合并*branchs*多个分支（如果冲突则不会执行）

当双方均有修改而出现冲突时，打开编辑器手动解决
git rebase --continue|abort|skip 确定工作流是继续（若已解决，需先用 **git add** 更新至索引，后 commit 或 --continue）还是中止（撤退回命令前状态）（skip 将跳过该节点的修改）
git merge --continue|abort 同上

远端（补充 **2**）：

git remote [-v] 查看所有已载远程仓库，-v 还可显示地址
git remote show *remotename* 显示某个远程仓库的详细信息
git remote add *remotename url* 添加远程仓库
（clone 默认仓库名为 origin，如果需要指定可以 **git clone -o *remotename url*** ）
git remote rm *remotename* 删除某远程仓库
git remote rename *oldname newname* 重命名远程仓库

其他：

git grep *pattern* 在当前工作树搜索指定文本，可以使用正则表达式（多行匹配模式）

git stash 将当前所有未提交的修改储藏（刻录暂存区与工作区的状态）
git stash save *message* 附上储藏信息
git stash list 列表显示已存的各次储藏
git stash show stash@{*num*} 指定某次储藏查看，**-p** 显示详细 diff，默认查看最近一次储藏
git stash pop 取出最近一次储藏，全部恢复到工作区（**--index** 可以把对应的暂存区内容恢复到暂存区）
git stash apply stash@{*num*} 应用某次储藏到工作
git stash drop stash@{*num*} 删去某次储藏，**git stash clear** 清空所有储藏

git config --local|global|system 显示配置（本仓>全局>系统，无指定则全部当前配置），**-l** 以列表简洁显示，**-e** 将用编辑器打开配置文件
git config --*location* --add *section.key value* 添加配置项，*location*默认为 local，--add 可以省略
git config --global alias.unstage 'reset HEAD --'
git config --*location* --unset *section.key* 删去某配置项

ssh-keygen -t rsa -C *accountemail* 在 GitHub 注册后生成 SSH key，将生成 .ssh 文件夹，内含公钥 is_rsa.pub 和私钥 is_rsa，复制公钥添加至 GitHub 账户，可输入 **ssh -T git@github.com** 测试是否成功

Git 会无视空文件夹（可以含空子文件夹）

*Hash*引用可以只输入（至少）前四位

gitk 打开 gitk 可视化窗口管理