第二章 git basic

1. git commit –a： 表示跳过 add 直接 commit
2. git rm
   a) git rm log/*.log：表示删除 log 文件夹下.log 结尾的文件
   b) git rm –f ：把已经放入 staged 的文件完全删除
   c) git rm --cached : 把已经 staged 的文件从暂存区清除，但是保留在工作区
   d) git mv file_from file_to：简便的让 git 追踪 rename 的方法
3. git log
   a) git log -p：表示列出每一个 commit 与上一个的详细差距
   b) git log --stat：表示列出每一个 commit 与上一个的简略差距
   c) git log --pretty=oneline：表示把每个 commit 缩短到一行显示。同时：short（显示 commit 和 author）full 还有 fuller 会显示日期
   d) git log --pretty=format:"%h - %an, %ar : %s"：表示以特定格式输出
   e) committer：提交者。Author：原作者

%H	Commit hash
%h	Abbreviated commit hash
%T	Tree hash
%t	Abbreviated tree hash
%P	Parent hashes
%p	Abbreviated parent hashes
%an	Author name
%ae	Author e-mail
%ad	Author date (format respects the --date=option)
%ar	Author date, relative
%cn	Committer name
%ce	Committer email
%cd	Committer date
%cr	Committer date, relative
%s	Subject

   f) git log --pretty=format:"%h %s" –graph：表示以图形显示
   g) git log -n：表示只显示最近的 n 个 commit
   h) git log --since=2.weeks/--until：表示时间上的 filter
   i) git log –author：搜索特定作者。--grep 表示在 commit message 里寻找特定字符串
   j) git log --all-match：表示用 and 连接所有的搜索限制条件

k) git log -Sfunction_name：-S 表示在更改的 code 里搜索特定字符串。

l) git log -- path：表示只寻找更改过特定文件的 commit

m) git log --decorate：表示显示 git 指针

n) git log --pretty="%h - %s" --author=gitster --since="2008-10-01" --before="2008-11-01" --no-merges：--no-merges 表示不要打印多于一个 parent 的 commit，等同于--max-parents=1。--merges 表示只打印 merges，等同于--min-parents=2

Option	Description
-p	Show the patch introduced with each commit.
--stat	Show statistics for files modified in each commit.
--shortstat	Display only the changed/insertions/deletions line from the --stat command.
--name-only	Show the list of files modified after the commit information.
--name-status	Show the list of files affected with added/modified/deleted information as well.
--abbrev-commit	Show only the first few characters of the SHA-1 checksum instead of all 40.
--relative-date	Display the date in a relative format (for example, “2 weeks ago”) instead of using the full date format.
--graph	Display an ASCII graph of the branch and merge history beside the log output.
--pretty	Show commits in an alternate format. Options include oneline, short, full, fuller, and format (where you specify your own format).

Option	Description
--author	Only show commits in which the author entry matches the specified string.
--committer	Only show commits in which the committer entry matches the specified string.
--grep	Only show commits with a commit message containing the string
-S	Only show commits adding or removing code matching the string

4. git undos

a) git commit --amend：用法是当 commit 完之后如果发现有一些文件需要改动或者没

有添加，那么可以先把文件 git add 进 staged 区，然后 git commit –amend。可以**修改上次的 message 和把这次 add 进来的文件提交到上一次 commit。**

b) git reset HEAD CONTRIBUTING.md：把已经 staged 的文件 unstage 或者已经 commit 的文件 uncommit，不修改文件内容，只修改文件跟踪记录。可以接具体文件名

c) git reset --hard：不能接文件名，可以接 HEAD^或者 commit 版本号，完全回退到指定 commit，放弃所有修改

d) git reset --soft: 不能接文件名，可以接 HEAD^或者 commit 版本号，回退到提交之前，不回退文件跟踪记录和文件内容，用 commit 可以直接提交

5. git checkout
   a) git checkout -- CONTRIBUTING.md：--和路径名之前有空格，表示把一个文件完全恢复到上次 commit 的状态，丢弃所有修改
6. git remote 相关
   a) git remote -v：显示更多信息
   b) git remote show [origin]：显示详细信息
   c) git fetch [origin]
   d) git pull 和 git fetch：前者自动把内容 merge 进来，后者不自动 merge
   e) git push [remote-name] [branch-name]
   f) git remote rename [new-name] [old-name] 远程分支重命名
   g) git remote rm [branch-name] 删除远程分支
7. git tag 相关
   a) git tag：表示列出当前所有 tag
   b) git tag -l/--list “pattern”：表示列出当前所有符合 pattern 的 tag
   c) git tag -d/--delete：表示删除对应标签
   d) lightweight tag：表示轻量级 tag，只是一个指向 commit 的指针
      git tag v1.4-lw：不加-a 表示这是一个轻量级 tag，只是一个 checksum，不含任何其他信息。
   e) annotated tag：完整的 tag，包含 message，email，author 等数据
      git tag -a [tag-name] -m 'my version 1.4'：-a=--annotate，表示新建一个没有符号的 annotated tag。-m=--message 表示 tag 的信息
   f) git show [tag-name] 表示显示这个 tag 的详细信息
   g) git tag (-a [tag-name]) [commit-hash]：表示给指定 commit 加上 tag
   h) git push [remote-name] [tag-name]：默认情况下，tag 并不自动 push 到远端 server。使用 git push 手动 push tag 到远端 repo. –tags 可以一次把所有的 tag 都 push 到远端服务器
8. git alias：用于定义 git 短命令
   a) git config –global alias.[shortcut] ‘[long command]’

第三章 git branching

9. git branch
   a) git branch [branch-name]：表示新建一个 branch，但是不自动切换到这个 branch 上
      git checkout -b [branch-name]：表示新建并切换到这个 branch 上

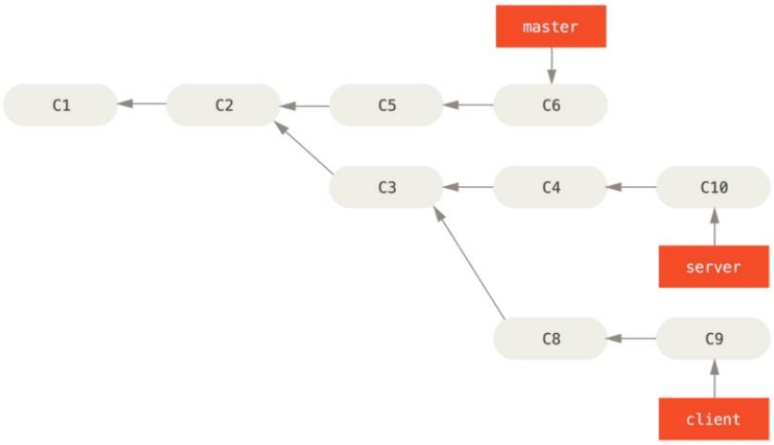
git checkout [branch-name]：表示切换到指定的 branch 上

b) git branch -d [branch-name]：表示删除某个 branch，通常在该 branch 已经被 merge 之后
c) git branch -v：显示 branch 的详细信息
d) git branch --merged/--no-merged：表示显示已经合并或者没有合并的分支

10. git merge
    a) git merge [branch]：分为三种情况
        i. 要 merge 进来的 branch 指针在本分支之前，这种情况只需要向前移动指针，是 fast-forward 模式
        ii. 要 merge 进来的 branch 指针与本分支均有 commit 但没有冲突，这种情况需要把要 merge 进来的分支里的改动加到本分支来，属于 recursive 模式
        iii. 要 merge 进来的 branch 指针与本分支均有 commit 且有冲突，这种情况需要修改冲突并重新 commit
    b) git merge --abort：表示如果 merge 过程中出现冲突，会直接回退到 merge 之前。否则，要手动使用 git reset --merge 来回滚
    c) git merge --no-commit --squash feature: 把 feature 上的所有内容合并到当前分支上，并且不添加新的 commit
11. git branch 使用风格
    a) 保持一条长期的 branch（dev 或者 next），在 dev 分支已经稳定的时候可以把它 merge 到 master 分支里
    b) 针对特定的 topic/issue 开新的短期分支，并且在 feature 解决后合并该分支到 dev 并且删除原来的分支
    c) 总是把某分支合并到更加稳定的分支上去
12. 远程分支
    a) git fetch [remote-name]：表示把远程相应分支的代码拉到本地，但是不主动 merge
       git merge [remote-name]/[remote-branch]：表示把 fetch 下来的代码合并
    b) git push [remote-name] [local-branch]:[remote-branch]：把代码 push 到远程特定 branch 上
    c) git push -f/--force：表示强行覆盖 server 端的历史，一般是本地 rebase 或者回滚之后要做的
    d) git checkout --track [remote-name]/[remote-branch]：新建一个同名 branch，并让这个 branch 跟踪远程分支
    e) git checkout -b [branch-name] [remote-name]/[remote-branch]：新建一个特定名字的 branch，并让这个 branch 跟踪远程分支
    f) git branch -u/--set-upstream-to [remote-name]/[remote-branch]：改变当前分值所跟踪的远程分支
    g) git push [remote-name] -d [remote-branch]：删除远程分支
13. git rebase：同 git merge 结果相同，是把【当前分支】的所有改动打成一个补丁，直到找到 common ancestor，之后把这个补丁打到【目标分支】上去。git rebase 会把历史变成线性，从而使 history 简短而清晰
    a) git rebase [basebranch] [topicbranch]：表示把 topic 分支做成补丁打到 base 上去。如果省略 topic，则表示把当前分支打到 base 上去
    b) git checkout [branch-name]
       git rebase master

git checkout master
git merge [branch-name]

c) 一个 rebase 案例



如果我想把 client 上的补丁打到 master，同时又保留 server 分支，应该这么做：
git rebase --onto master server client
意思是：Check out the client branch, figure out the patches from the common ancestor of the client and server branches, and then replay them onto master
之后把 server 补丁打到 master 上
git rebase master server

d) Rebase 原则：**Do not rebase commits that exist outside your repository.**
也就是说，不要 rebase 公共分支，这样会导致其他人 pull 下来的分支历史混乱

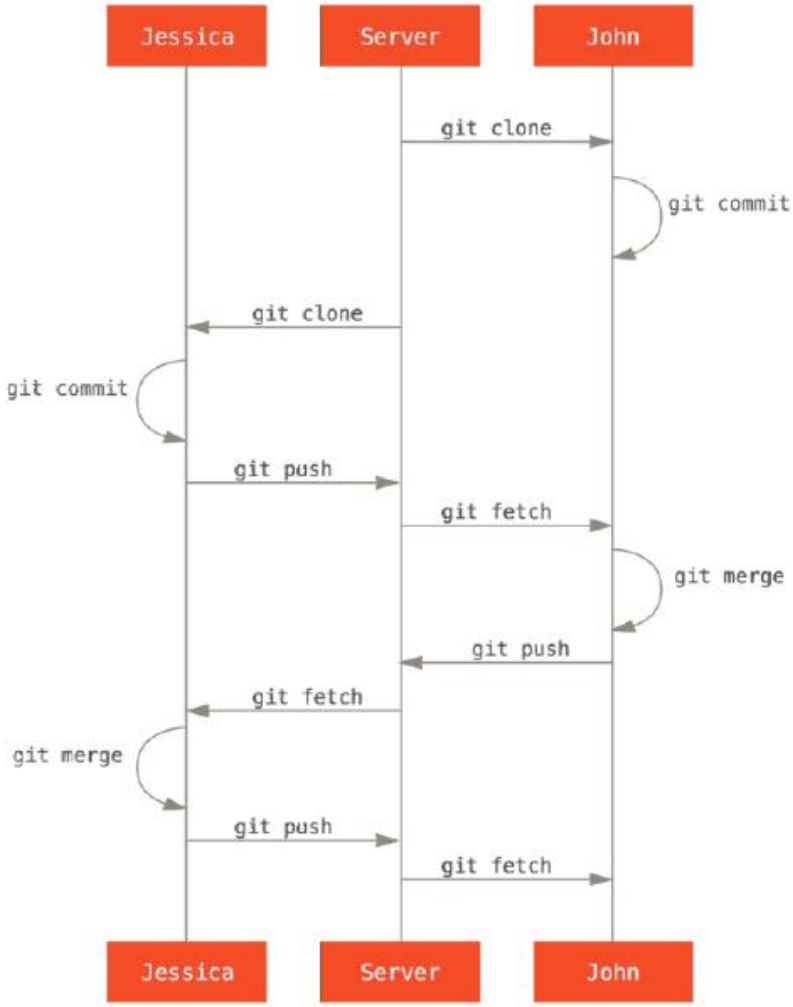
e) Git 对 rebase 有特殊处理：如果 merge 后生成的 commit 与 rebase 生成的 commit 非常相似，那么 git 会自动在本地 pull 下 rebase 后的分支。或者也可以使用
git pull --rebase 来拉取被 rebase 的公开分支

14. Rebase vs merge：关键问题：git 仓库是**记录所有真实发生的（应该使用 merge）**还是**你的 project 是怎么做出来的（应该使用 rebase）**

第四章 git on the server

1. The protocols
   a) Local protocol: 本地文件系统或者网络文件系统
   git clone /opt/git/project.git（比较快，需要本地硬盘文件）
   git clone file:///opt/git/project.git（相对慢）
   b) http/s 协议：http 协议的好处主要在于不需要预先把公钥传给服务器，而且对防火墙友好
   c) ssh 协议
   git clone ssh://user@server/project.git 或者
   git clone user@server:project.git
   ssh 简单易执行，但是无法大面积共享程序
   d) git 协议：最快的协议，没有任何认证，需要 git-daemon-export-ok 文件。配置复杂，对防火墙不友好

2. getting git on a server
   a) export an existing repo into a new bare repo
      git clone --bare my_project my_project.git
      git init --bare --shared
      --shared 会自动把写权限加入 repo 中
      别人想 clone 这个 repo 的时候（如果已经把 ssh 公钥放入认证 key 中）
      git clone user@git.example.com:/opt/git/my_project.git
   b) ssh 接入
      ssh-keygen
      cat ~/.ssh/id_rsa.pub
      cat /tmp/id_rsa.john.pub >> ~/.ssh/authorized_keys
      使用 chsh 把 git-shell 变成 repo 使用者的登陆 shell。这个 shell 只能用来执行 git 的相关操作，不能登录进 linux 系统
   c) git daemon
      git daemon --reuseaddr --base-path=/opt/git/ /opt/git/
      --reuseaddr：允许 server 直接启动而不需要等待旧连接断掉
      --base-path：使用部分路径、
      必须把 9418 端口打开留着给 git 来用
   d) 加入自动启动进程：
      start on startup
      stop on shutdown
      exec /usr/bin/git daemon \
       --user=git --group=git \
       --reuseaddr \
       --base-path=/opt/git/ \
       /opt/git/
      Respawn
      直接启动 daemon：initctl start local-git-daemon
      最后在 git repo 里允许未认证 git 操作：touch git-daemon-export-ok
   e) Smart HTTP 接入
3. Distributed Git
   a) Github 式 workflow
      i. The project manager pusher their repos
      ii. A contributor clones this repo and makes change.
      iii. The contributor pushes to their own repos
      iv. Send email/pull request
   b) Commit guideline:
      i. Write useful commit messages.
      ii. One Commit pre issue
   c) Git request-pull:生成一个 pull 的 summary，等于无 github 版的 pull request
   d) 如果 branch 被 rebase 过，那 push 的时候应该加-f



第五章 git 高级命令

1. git rev-parse [branch-name]: 显示当前分支的当前 commit 的 hash 值
2. git reflog: 显示 HEAD 指针的移动路径，所有的 reflog 都是本地的
3. git show HEAD@{5}: 显示某个 commit 的详细信息
   a) git show master@{yesterday}: 显示昨天提交的所有 commit
   b) git show d921970^：显示某个 commit 的详细内容，^表示向前一个
   c) git log -g: 显示 reflog 相似的内容
4. 双点操作符..
   a) git log master..experiment：显示所有在 experiment 上但不是在 master 上的 commit
   b) git log refA..refB
      git log ^refA refB
      git log refB --not refA
      以上所有的写法是一样的，但是不用..可以用多个分支
      git log refA refB ^refC
      git log refA refB --not refC

5. 三点操作符...
   a) git log master...experiment: 显示所有被两个分支任意一个 access，但不能被两个分支同时 access 的 commit
6. 交互式 staging
   a) git add -i：

```
F:\University\CSCI4140\asgn2 [master +0 ~1 -0]> git add -i
           staged     unstaged path
  1:         +6/-0      nothing views/index.html

*** Commands ***
  1: status       2: update       3: revert       4: add untracked
  5: patch        6: diff         7: quit         8: help
What now>
```

      i. update: 使用序号添加文件，然后敲回车，选中的文件会被 staged
      ii. revert：使用序号添加文件，然后敲回车，选中的文件会被 upstaged
      iii. diff：使用序号添加文件，然后敲回车，选中的文件会显示 diff。等于 diff --cached

   b) git add -p: 制作可以用来打 patch 的文件
7. git stash
   a) git stash: 保存当前所有改动，并把目前目录重设到上一个 commit
   b) git stash list：显示目前所有 stash 记录
   c) git stash apply：弹出栈顶的 stash 记录并把该记录 apply 到当前分支。可以 apply 到其他分支
   d) git stash apply stash@{2}：弹出指定的 stash 记录并把该记录 apply 到当前分支
   e) git stash apply --index：把已经 staged 的文件全部 unstage 回去
   f) git stash drop stash@{0}：删除指定的 stash 记录
   g) git stash --keep-index：把已经 staged 的文件也都 stash 掉
   h) git stash -u/--include-untracked：把 untracked 的文件也一起 stash 掉
   i) git stash --patch：显示一个 patch，同时 stash
   j) git stash branch [branch-name]：stash 之后再恢复，可能出现 confilct。此时应该开一个新分支，stash 的内容 apply 上去，然后使用 git merge
   k) git stash --all：把所有当前的内容完全保存并清空
8. git clean
   a) git clean -d：删除所有 untracked 的文件。**不可恢复。**
   b) git clean -f/-n：表示实际删除/显示将会删除什么内容
   c) git clean -x：同时删除在 gitignore 里的文件
   d) git clean -i：交互式 git clean

```
Would remove the following items:
  .idea/.name                   .idea/misc.xml
  .idea/jsLibraryMappings.xml   .idea/workspace.xml
*** Commands ***
    1: clean                2: filter by pattern    3: select by numbers
    4: ask each             5: quit                 6: help
What now>
```

9. git grep
   a) git grep -n [string]：显示当前文件里所有含有 string 的行。-n 表示显示行号
   b) git grep --count [string]：显示当前文件里含有 string 的行数

c) git grep -p [string] [file range]：表示寻找这个 string 在文件里的某个函数
d) 比 grep 和 ack 的优势
   i. 快
   ii. 可以沿着任何一个 commit 搜索，而不只是当前目录。

10. git log 其他用法
a) git log -S[string] --oneline：表示显示把[string]引入/删除的 commit
b) git log -G[regex]：提供一个正则表达式
c) git llog -L :[function_name]:[file_name]:表示在一个文件里寻找一个函数，然后列出这个函数所有的历史

11. 改变历史
a) git commit --amend：
   i. 更改 commit message。
   ii. 如果想更改文件，先做 git add/rm，然后再 git commit --amend。Git 会自动把当前的 stage 区域更新原来的。
b) Git rebase -i：commit 是以正序排列，**不能使用在已经 push 到仓库的 commit**
   i. 后面接 HEAD^n，用于同时修改多个 commit

```
pick f7f3f6d changed my name a bit
pick 310154e updated README formatting and added blame
pick a5f4a0d added cat-file

# Rebase 710f0f8..a5f4a0d onto 710f0f8
#
# Commands:
#  p, pick = use commit
#  r, reword = use commit, but edit the commit message
#  e, edit = use commit, but stop for amending
#  s, squash = use commit, but meld into previous commit
#  f, fixup = like "squash", but discard this commit's log message
#  x, exec = run command (the rest of the line) using shell
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
```

   ii. 可以修改 commit 的顺序，git 会依次打补丁
   iii. 可以把多个 commit 合成一个
   iv. 可以分裂某一个 commit。具体做法
      1. 把 pick 改成 edit
      2. 回到 console 之后，使用 git reset HEAD^来重新编辑 commit
      3. 提交之后，输入 git rebase --continue 继续 rebase
c) 改变大量历史：filter-branch。一般用在需要修改 email 等地方，**会改变所有的 sha-1，不能用在已经 push 到仓库的 commit 上。**
   i. git filter-branch --tree-filter 'rm -f passwords.txt' HEAD：遍历当前分支里所有的

节点，对每一个节点执行命令，然后再 commit。加上-all 表示应用在所有的 commit 上

ii. git filter-branch --subdirectory-filter trunk HEAD：将 trunk 设置为当前分支的 root，git 会自动删除所有不含 trunk 文件夹改动的 commit

iii. 脚本，修改全局 Email

```
git filter-branch --commit-filter '
    if [ "$GIT_AUTHOR_EMAIL" = "schacon@localhost" ];
    then
        GIT_AUTHOR_NAME="Scott Chacon";
        GIT_AUTHOR_EMAIL="schacon@example.com";
        git commit-tree "$@";
    else
        git commit-tree "$@";
fi' HEAD
```

12. 高级 reset
    a) git ls-tree -r HEAD：显示 HEAD 指针的路径
    b) git reset 第一步：移动 HEAD 指针
    c) git reset [sha-1] -- [file-path]：不移动 HEAD 指针，不改变工作区文件，**只把指明的 commit 里的 file 复制出来并放在当前 index 里。**
    d) 挤压 commit：首先使用 git reset --soft HEAD^n 回到 n 个 commit 之前，然后直接 commit 当前文件，即可将 n 个 commit 挤压成 1 个 commit

13. Reset VS checkout
    a) Checkout -- [file_name]时，工作区的文件会被覆盖
       Reset -- [file_name]时，工作区的文件不会被修改
    b) Checkout [branch]时，master 指针不动，branch 指针不动，HEAD 指针移动
       Reset [branch]时，master 指针不动，branch 和 HEAD 指针同移动到 master 指针上

14. 高级 Merge
    a) 当 merge 遇到 conflict 时，应该使用 git merge --abort 回滚 merge 操作。不能有未 track 或者 stage 的文件
    b) git merge -Xignore-all-space：表示 merge 的时候忽略所有空格引起的冲突
    c)