## Git常用命令

* **创建版本库**

初始化本地版本库：git init

克隆远程版本库：git clone <url>

* **远程仓库操作**

连接远程版本库git remote add origin [git@github.com:caor22/hello.git](mailto:git@github.com:caor22/hello.git→)

抓取数据合并到本地git pull / git pull <remote> <branch>

推至远程版本库git push origin master / git push（第1次git push -u origin master）

查看远程版本库信息 git remote -v

查看指定远程版本库信息 git remote show <remote>

从远程仓库抓取数据（本地仓库中还没有的数据）git fetch <remote> <branch>【 git pull = git fetch + git merge】

从远程仓库的分支仓库抓取数据到新分支下git fetch <remote> <branch>：<new\_branch>

添加远程仓库git remote add [shortname] [url]

移除远程仓库git remote rm [shortname]

* **修改和提交**

工作区（work）→暂存区（staged/index）→版本库（git）

查看仓库状态：git status

添加到暂存区：git add <file>

跟踪所有改动过的文件 git add .

提交到版本库:git commit -m”<message>”

越过暂存区提交到版本库git commit -a -m”<message>”

修改最后一次提交 git commit --amend

移除版本区文件git rm <file>

强制删除工作区、暂存区文件git rm -f <file>

停止跟踪但不删除git rm --cached <file>

文件改名git mv <old> <new>

* **差异比较**

查看变更内容（工作区→暂存区）git diff

查看变更内容（暂存区→版本库）git diff --staged/cached

查看变更内容（工作区→版本库）git diff HEAD -- <file>

* **查看提交历史**

显示提交日志（穿梭过去）git log

查看所有提交历史版本号（重返未来）git reflog（p 选项展开显示每次提交的内容差异，用 -2 则仅显示最近的两次更新）

获取单词层面上的对比git log -p --word-diff

单词层面的对比（1行显示）git log -U1 --word-diff

不同于默认格式的方式展示提交历史（1行显示）$ git log --pretty=oneline

定制要显示的记录格式git log --pretty=format:"%h - %an, %ar : %s"

简单图形，分支及其分化衍合情况git log --pretty=format:"%h %s" --graph

* **撤消**

取消已经暂存的文件（不改文件）git reset HEAD <file>

取消对文件的修改（改文件）git checkout -- <file>

回滚版本git reset --hard <commit id>或git reset HEAD^（^上一版本）或git reset HEAD~100（上100版本）

撤销指定提交git revert <commit>

* **分支管理**

查看分支：git branch , git branch <-v,--merged,--no-merged>

查看远程分支： origin/master

同步远程服务器上的分支数据到本地git fetch origin

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>

合并远程分支到本地分支git merge origin/<name>

分支变基：git rebase <name> ，git rebase [主分支] [特性分支]

3分支选2分支rebase ：git rebase --onto master server client

删除分支：git branch -d <name>

强制禁用Fast forward模式，生成新的commit： git merge --no-ff -m "merge with no-ff" dev

跟踪分支git checkout --track origin/<name>

推送本地分支至远程版本库git push origin <name>

删除远程分支：git push [远程名] :[分支名]

获取所有远程索引 git pull

* **标签管理**

查看所有标签git tag

创建标签git tag <name>或 git tag <name> <commit id>

指定提交信息git tag -a <tagname> -m”<message>”

查看标签内容git show <tagname>

删除本地标签git tag -d name

删除远程标签git push origin :refs/tags/v0.9git

发表标签git push origin <tagname>

发表所有标签git push origin --tags

切换标签：git checkout <name>

验证已经签署的标签 git tag -v [tag-name]

* **其他**

显示当前目录pwd

展示下面所有的文件ll

到上一级目录cd..

看到.git目录中的文件和子目录Ls

输出"写入内容"追加到>><file> echo "写入内容" >> <file>

展示文件内容cat <file>

清除工作区，未跟踪文件git clean

储藏git stash

查看现有的储藏git stash list

重新应用你刚刚实施的储藏git stash apply

重新应用stash@{2}储藏git stash apply stash@{2}

对文件的变更被重新应用，但是被暂存的文件没有重新被暂存git stash apply --index

移除储藏git stash drop stash@{2}

重新应用储藏git stash pop

 取消储藏stash unapply

从储藏中创建分支 git stash branch

Git 基础 - 技巧和窍门

自动补全<tab>

匹配的可用命令建议<tab><tab>

* **场景**

-保存和恢复进度

保存和恢复进度git stash修复后，git stash pop，回到工作现场。

工作现场还在，Git把stash内容存在某个地方了，但是需要恢复一下，有两个办法：一是用git stash apply恢复，但是恢复后，stash内容并不删除，你需要用git stash drop来删除；另一种方式是用git stash pop，恢复的同时把stash内容也删了；

-多人协作的工作模式

首先，可以试图用git push origin <branch-name>推送自己的修改；

如果推送失败，则因为远程分支比你的本地更新，需要先用git pull试图合并；

如果合并有冲突，则解决冲突，并在本地提交；

没有冲突或者解决掉冲突后，再用git push origin <branch-name>推送就能成功！

如果git pull提示no tracking information，则说明本地分支和远程分支的链接关系没有创建，用命令git branch --set-upstream-to <branch-name> origin/<branch-name>。

* **GitHub**

[派生项目](https://git-scm.com/book/zh/v1/%E6%9C%8D%E5%8A%A1%E5%99%A8%E4%B8%8A%E7%9A%84-Git-Git-%E6%89%98%E7%AE%A1%E6%9C%8D%E5%8A%A1" \l "%E6%B4%BE%E7%94%9F%E9%A1%B9%E7%9B%AE)：要为一个自己没有推送权限的项目贡献代码，GitHub 鼓励使用派生（fork）。

* **我的信息**

用户名：caor22

密码：Cr18811657411

hello\_ssh\_key：9e:c7:1b:b3:09:14:e2:7c:e6:9f:9a:1b:ed:54:bd:0b

...或在命令行上创建一个新的存储库

echo“＃hello2”>> README.md

git init

git add README.md

git commit -m“first commit”

git remote add origin https://github.com/caor22/hello2.git

git push -u origin master

...或从命令行推送现有存储库

git remote add origin https://github.com/caor22/hello2.git

git push -u origin master

...或从另一个存储库导入代码

您可以使用Subversion，Mercurial或TFS项目中的代码初始化此存储库。

* **附录**

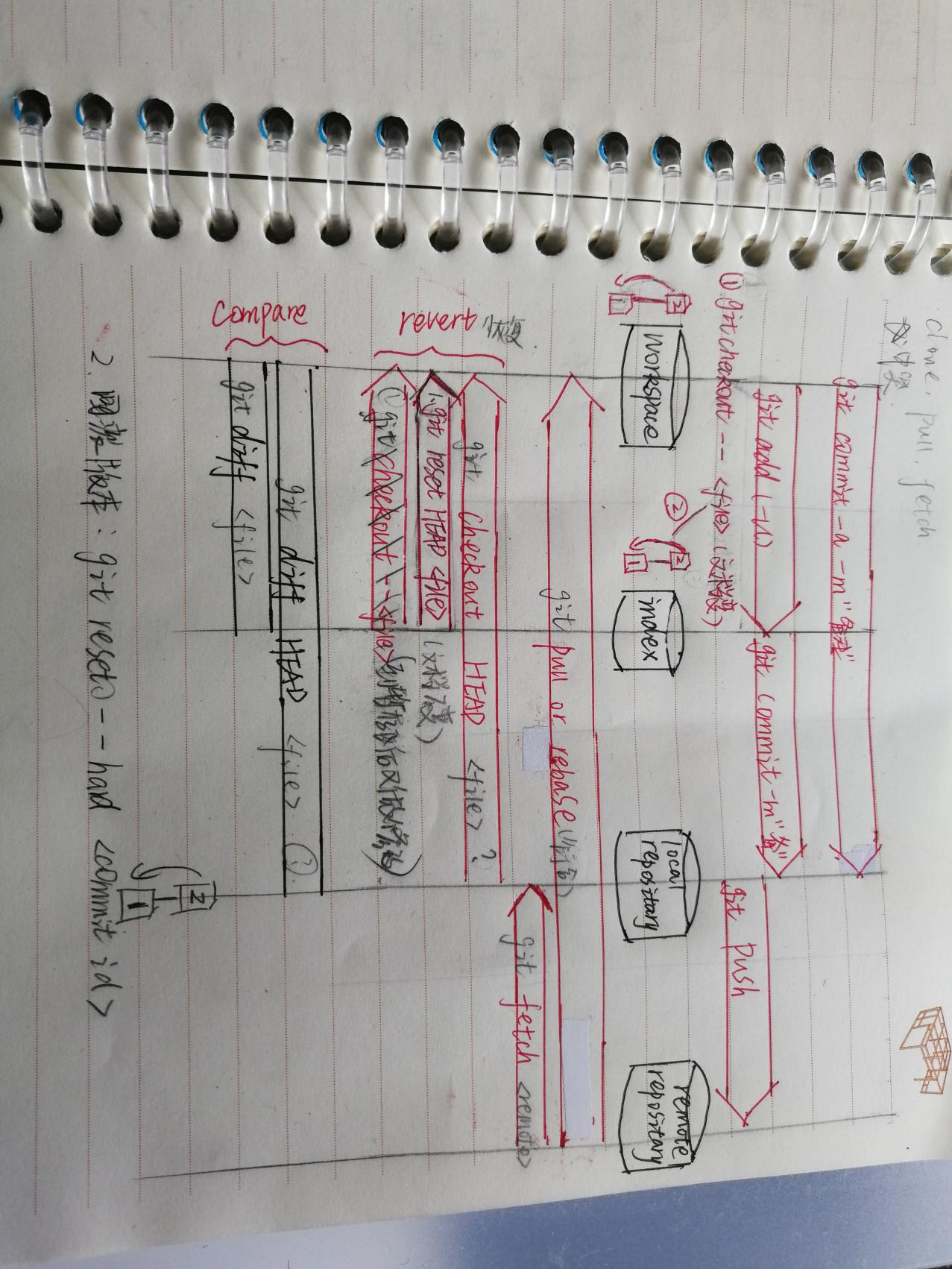
表 2-1 列出了常用的格式占位符写法及其代表的意义。

| **选项** | **说明** |
| --- | --- |
| %H | 提交对象（提交）的完整哈希字串 |
| %h | 提交对象的简短哈希字串 |
| %T | 树对象（树）的完整哈希字串 |
| %t | 树对象的简短哈希字串 |
| %P | 父对象（父）的完整哈希字串 |
| %p | 父对象的简短哈希字串 |
| %an | 作者（作者）的名字 |
| %ae | 作者的电子邮件地址 |
| %ad | 作者修订日期（可以用-date =选项定制格式） |
| %ar | 作者修订日期，按多久以前的方式显示 |
| %cn | 提交者（提交者）的名字 |
| %ce | 提交者的电子邮件地址 |
| %cd | 提交日期 |
| %cr | 提交日期，按多久以前的方式显示 |
| %s | 提交说明 |

表2-2还列出了一些其他常用的选项及其释义。

| **选项** | **说明** |
| --- | --- |
| -p | 按补丁格式显示每个更新之间的差异。 |
| --word-diff | 按字diff格式显示差异。 |
| --stat | 显示每次更新的文件修改统计信息。 |
| --shortstat | 只显示--stat中最后的行数修改添加移除统计。 |
| --name-only | 仅在提交信息后显示已修改的文件清单。 |
| --name-status | 显示新增，修改，删除的文件清单。 |
| --abbrev-commit | 仅显示SHA-1的前几个字符，而非所有的40个字符。 |
| --relative-date | 使用较短的相对时间显示（比如，“2周前”）。 |
| --graph | 显示ASCII图形表示的分支合并历史。 |
| --pretty | 使用其他格式显示历史提交信息。可用的选项包括oneline，short，full，fuller和format（后跟指定格式）。 |
| --oneline | --pretty=oneline --abbrev-commit 的简化用法。 |

文档版本管理调研和分享

不懂：

1、hello第二次push>>有冲突提交不了

2、克隆，怎么复制远程已有内容的仓库url源路径>>用github上的Web URL

3、github 分支图

4、clone失败>>用github上的Web URL

5、bug分支stash

6、Merge和rebase适应场景：

****一旦分支中的提交对象发布到公共仓库，就千万不要对该分支进行变基操作。****

7、

8、