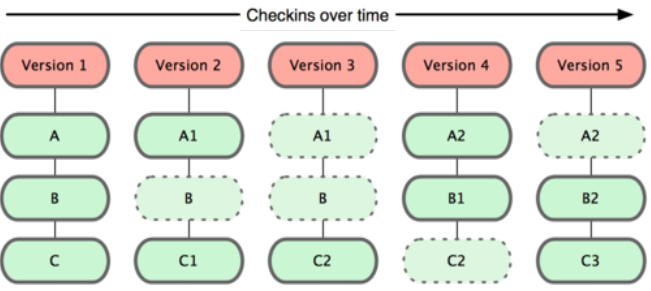
Git常用指令整理

# Git特点

* 直接快照，而非差异比较

Git和其它版本控制系统的主要差别在于，Git只关心文件数据的整体是否发生变化。Git并不保存这些前后变化的差异数据，实际上，Git更像是把变化的文件作快照后，记录在一个微型的文件系统中。每次提交更新时，它会纵览一遍所有的文件的指纹信息并对文件作一快照，然后保存一个指向这次快照的索引。为提高性能，若文件没有变化，Git不会再次保存，而只对上次保存的快照作一连接。Git的工作方式如下图所示：



* 近乎所有操作都可本地执行

在Git中的绝大多数操作都只需要访问本地文件和资源，不用连网。就算在飞机或者火车上，都可以非常愉快地频繁提交更新，等到了有网络的时候再上传到远程的镜像仓库。

* 时刻保持数据完整性

如果文件在传输时变得不完整，或者磁盘损坏导致文件数据缺失，Git都能立即察觉。

* Git常用操作：



# 克隆已有仓库

$ git clone [git@github.com:XXX/YYY.git](mailto:git@github.com:XXX/YYY.git) newprj\_name

XXX：github的用户名

YYY：仓库名

# 隐藏修改

Git stash

Git stash list

Git stash pop

# Help

$ git help <verb>

$ git <verb> --help

# 增加/删除文件

* 添加指定文件到暂存区

$ git add [file1] [file2] …

* 添加指定目录到暂存区，包括子目录

$ git add [dir]

* 添加当前目录的所有文件到暂存区

$ git add .

* 同时删除工作目录和暂存区的文件

$ git rm [file1] [file2] …

注意：$ rm test.txt 仅删除工作目录中的文件，暂存区的文件还在

* 仅删除暂存区的文件，工作目录的文件还在

$ git rm --cached [file]

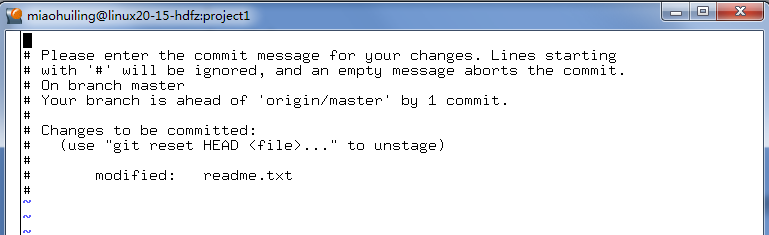
* 文件改名，并将这个改名放入暂存区

$ git mv [file-original] [file-renamed]

# 代码提交

* 文本编辑器提交暂存区到仓库区：在光标处输入提交信息

$ git commit



* 命令行提交暂存区到仓库区

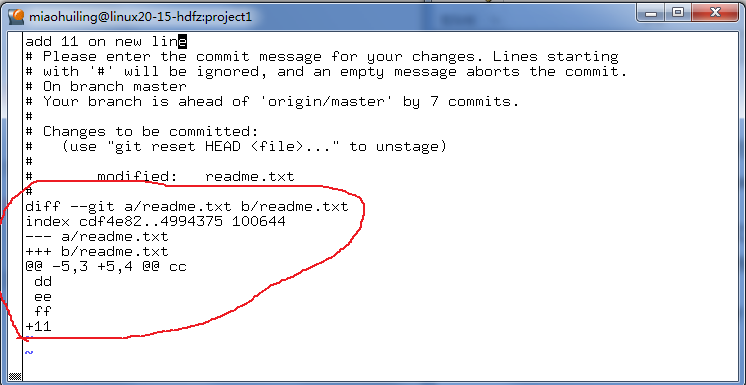
$ git commit –m “ update message xxx”

* 提交暂存区的特定文件到仓库区

$ git commit [file1] [file2] … –m “ update message xxx”

* 提交时显示所有diff信息

$ git commit –v



* 跳过使用暂存区域：给git commit 加上 –a 选项，git会自动把所有已经跟踪过的文件暂存起来一并提交，从而跳过 git add 步骤。
* 使用暂存区域的步骤：

$ git add xxx

$ git commit –m “update”

* 跳过暂存区域：

$ git commit –a –m “update xxx”

# 查看信息

* 显示有变更的文件

$ git status

* 使用图形化工具查阅提交历史

$ gitk

* 显示当前分支的版本历史
* 使用图形化工具查阅提交历史

$ gitk

* 按提交时间列出所有更新，最近的更新排在最上面

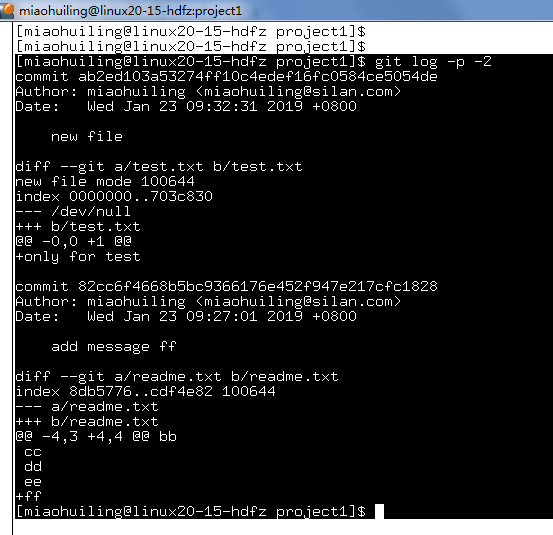
$ git log

* 显示每次提交的内容差异

$ git log –p -n “-p”展开显示每次提交的内容差异；

“-n”仅显示最近n次更新

例如：$ git log –p -2

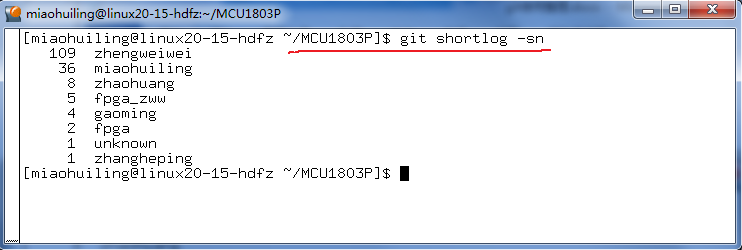


* 显示commit历史，以及每次commit发生变更的文件

$ git log --stat

* 显示所有提交过的用户，按提交次数排序

$ git shortlog –sn



* 显示工作区和暂存区的代码差异（不是这次工作和上次提交之间的差异）

$ git diff

* 显示暂存区和上一个commit的差异

$ git diff --cached [file]

* 显示工作区与当前分支最新commit之间的差异

$ git diff HEAD

# 远程同步

* 下载远程仓库的所有变动到本地仓库

$ git fetch [remote]

* 取回远程仓库的变化到本地（工作区）

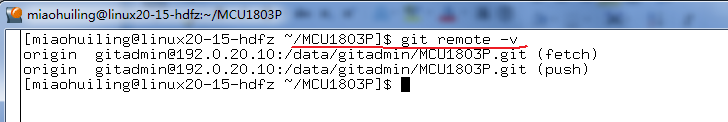
$ git pull [remote] [branch]

* 上传数据到远程仓库

$ git push [remote] [branch]

* 查看所有远程仓库的信息

$ git remote –v



* 显示某个远程仓库的信息

$ git remote show [remote]

# 撤销

* 取消对本地文件的修改，也既恢复暂存区的制定文件到工作区

$ git checkout [file]

* 恢复暂存区的所有文件到工作区

$ git checkout .

* 重置暂存区与工作区，与上一次commit保持一致

$ git reset --hard

# 分支

* 列出所有本地分支

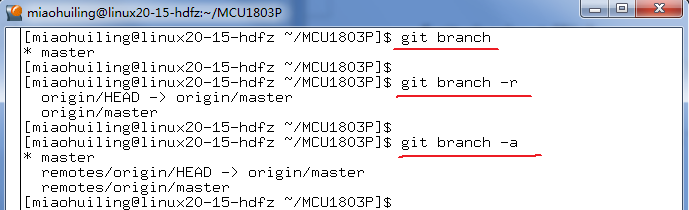
$ git branch

* 列出所有远程分支

$ git branch -r

* 列出所有本地分支和远程分支

$ git branch -a



* 新建一个分支，但依然停留在当前分支

$ git branch [branch-name]

* 新建一个分支，并切换到该分支

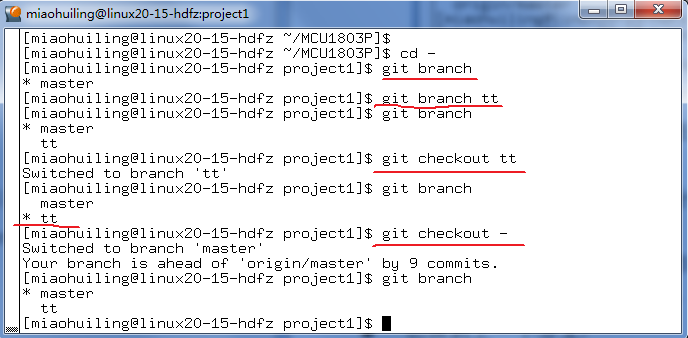
$ git checkout –b [branch]

* 切换到指定分支，并更新工作区

$ git checkout [branch-name]

* 切换到上一个分支

$ git checkout –



* 合并指定分支到当前分支

$ git merge [branch]

* 当前分支提交信息衍合到指定分支

Git rebase branch

* 删除分支

$ git branch –d [branch-name]

**.删除本地的远程分支：**

git branch -r -D origin/BranchName

**4.远程删除git服务器上的分支：**

git push origin -d BranchName

其中-d也可以是--delete，如：

git push origin --delete BranchName

**注意：git命令区分大小写，例如-D和-d在不同的地方虽然都是删除的意思，并且它们的完整写法都是--delete，但简易写法用错大小写会执行失败。**

# 标签

* 新建标签（含附注annotated的标签）

$ git tag –a xxx –m “my version 1.4”

* 列出所有标签

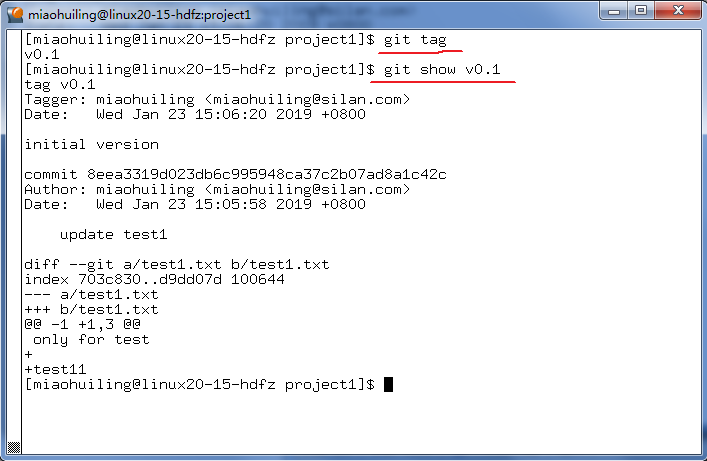
$ git tag

* 删除本地标签

$ git tag –d [tag]

* 查看tag信息

$ git show [tag]



* 提交指定tag

$ git push [remote] [tag]

* 删除远程tag：通过将远程分支合并进远程的master来实现

$ git push [远程名] [本地分支]:[远程分支]

# 管理项目子模块Submodule

Git Submodule允许一个Git仓库作为另一个Git仓库的子目录。并且保持父项目和子项目的相互独立。

使用项目子模块后，不必负责子模块的维护，只需要在必要的时候同步更新子模块即可。

1. **在父项目中添加项目子模块submodule**

进入到父项目的跟目录下，执行命令：

$ git submodule add [url] [path]

eg: $ git submodule add <https://gitee.com/moduleA.git> ./design/moduleA

用git status 可以看到多了两个需要提交的文件：moduleA 和 .gitmodules

在文件.gitmodule中可以看到url和path信息。

* **提交子模块信息到父项目的远程仓库**

$ git add .

$ git commit –m “add submodule A xxxx”

$ git push origin master

1. **克隆含有子仓库的项目**
   * 克隆一个包含子仓库的仓库目录，并不会clone下子仓库的文件，只是会克隆下.gitmodule描述文件，需要进一步克隆子仓库文件。

**方法一**：先clone父项目，再初始化submodule，最后更新submodule

|  |
| --- |
| $ git clone project.git project2  克隆完父项目后，可看到父项目下的submodule目录是空的。  $ cd project2  $ git submodule init # 初始化本地配置文件 .git/config  $ git submodule update # 更新子模块 |

**方法二：**采用递归 ”--recursive”

$ git clone project.git project3 --recursive

* + 上述操作都做完后，可以看到子目录在一个未命名分支，此时子仓库有改动的话检测不出来。



在子仓库，切换到master分支，并git pull最新代码之后，回到主仓库目录，会显示子仓库的修改状态。

1. **更新子模块**

先pull父项目，然后更新子模块

$ cd 父项目目录

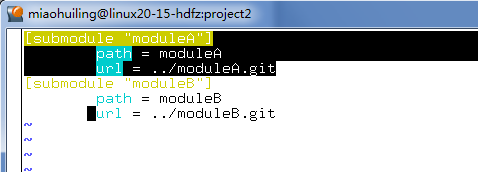
$ git pull

$ git submodule update

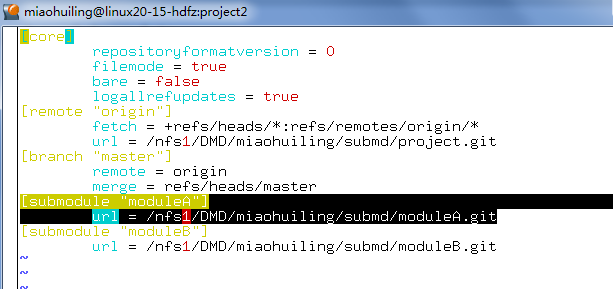
$ git submodule foreach git pull

1. **删除子模块**

* 删除暂存区的子模块文件：$ git rm --cached moduleA
* 删除子模块目录：$ rm –rf moduleA
* 删除.gitmodules中子模块的相关部分



* 删除 .git/config 中子模块的相关内容：



* 之后提交到远程服务器：

$ git add .

$ git commit –m “remove submodule”

1. **修改子仓库**

修改子仓库之后只对子仓库的版本库产生影响，对主仓库的版本库不会产生影响。如果在本地修改了子仓库，需要先将子仓库提交，然后再到主仓库提交代码。

|  |
| --- |
| # 子仓库提交  $ cd project/moduleA  $ echo “This is a submodule.” > b.txt  $ git add .  $ git commit –m “add b.txt”  $ git push origin master  $ cd ..  # 主仓库提交  $ git status  $ git add .  $ git commit –m “update submodue A add b.txt”  $ git push origin master |