Git初级技术分享

by SpeedX Dev. Team
2016年4月2日，Beijing

目录

什么是Git?

Git(分布式版本控制系统)

Git是一款免费、开源的分布式版本控制系统，用于敏捷高效地处理任何或小或大的项目。Git的读音为/gɪt/。

Git --- The stupid content tracker, 傻瓜内容跟踪器。Linus Torvalds 是这样给我们介绍 Git 的。

Git 是用于 Linux内核开发的版本控制工具。与常用的版本控制工具 CVS, Subversion 等不同，它采用了分布式版本库的方式，
不必服务器端软件支持（wingeddevil注：这得分是用什么样的服务端，使用http协议或者git协议等不太一样。并且在push和pull的时候和服务器端还是有交互的。），使源代码的发布和交流极其方便。
 Git 的速度很快，这对于诸如 Linux kernel 这样的大项目来说自然很重要。 Git 最为出色的是它的合并跟踪（merge tracing）能力。

实际上内核开发团队决定开始开发和使用 Git 来作为内核开发的版本控制系统的时候，世界开源社群的反对声音不少，
最大的理由是 Git 太艰涩难懂，从 Git 的内部工作机制来说，的确是这样。但是随着开发的深入，
Git 的正常使用都由一些友好的脚本命令来执行，使 Git 变得非常好用，即使是用来管理我们自己的开发项目，
Git 都是一个友好，有力的工具。现在，越来越多的著名项目采用 Git 来管理项目开发.

作为开源自由原教旨主义项目，Git 没有对版本库的浏览和修改做任何的权限限制。

简单的说，git就是在团队项目开发中，版本控制和代码控制的有利工具。

我们可以用Git来做什么?

开发成员:

1、从服务器上克隆完整的Git仓库（包括代码和版本信息）到单机上。
2、在自己的机器上根据不同的开发目的，创建分支，修改代码。
3、在单机上自己创建的分支上提交代码。
4、在单机上合并分支。
5、把服务器上最新版的代码fetch下来，然后跟自己的主分支合并。
6、生成补丁（patch），把补丁发送给主开发者。
7、看主开发者的反馈，如果主开发者发现两个一般开发者之间有冲突（他们之间可以合作解决的冲突），就会要求他们先解决冲突，然后再由其中一个人提交。如果主开发者可以自己解决，或者没有冲突，就通过。
8、一般开发者之间解决冲突的方法，开发者之间可以使用pull 命令解决冲突，解决完冲突之后再向主开发者提交补丁。

开发管理者:

1、查看邮件或者通过其它方式查看一般开发者的提交状态。

2、打上补丁，解决冲突（可以自己解决，也可以要求开发者之间解决以后再重新提交，如果是开源项目，还要决定哪些补丁有用，哪些不用）。

3、向公共服务器提交结果，然后通知所有开发人员。

git fetch：相当于是从远程获取最新版本到本地，不会自动merge

git pull：相当于是从远程获取最新版本并merge到本地

git merge 用来做分支合并，将其他分支中的内容合并到当前分支中。

以上三条指令作为了解，作为初学者后面git pull 用得较多

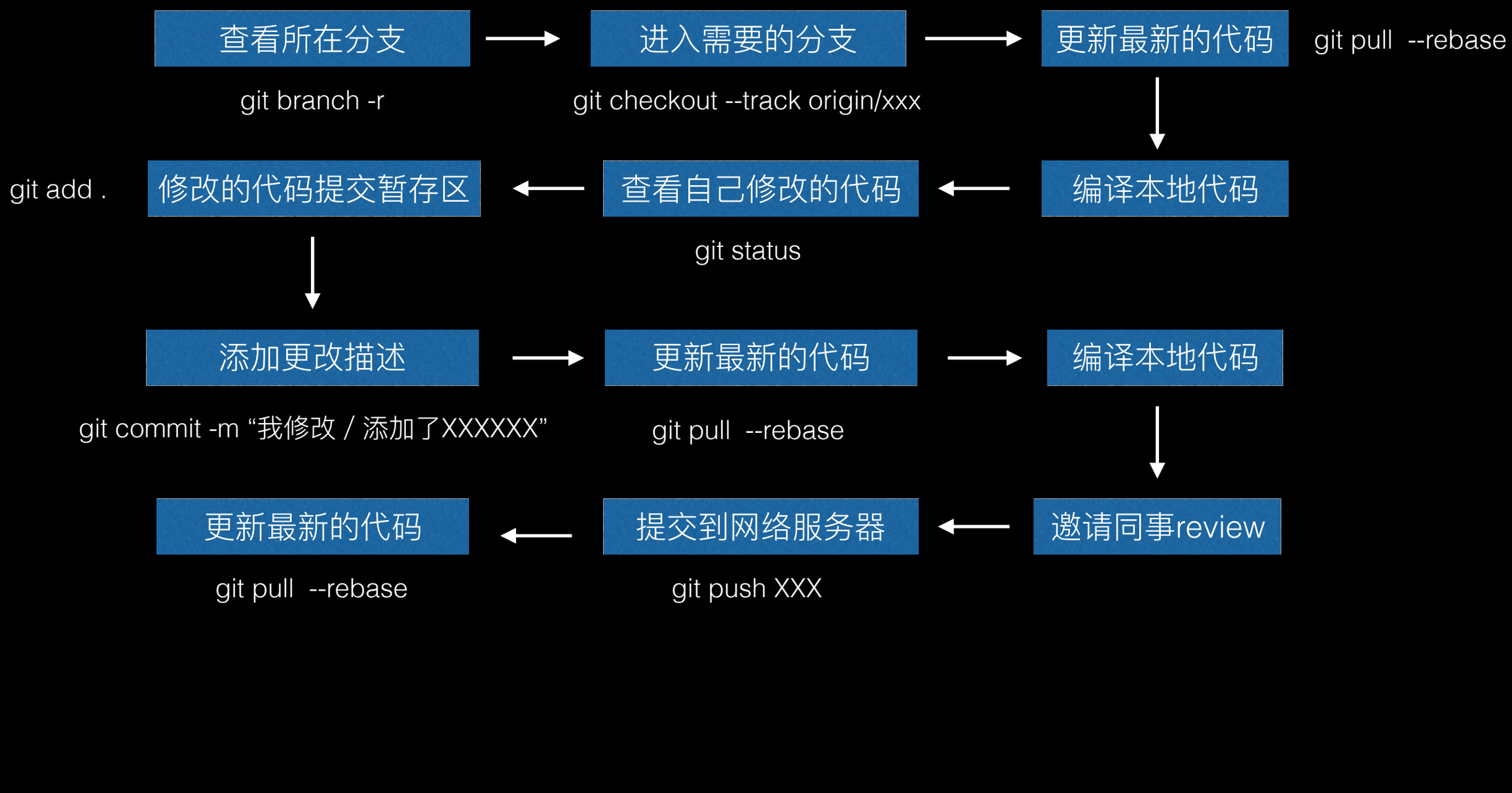
新手怎样快速入门?

- 准备工作：
  1. 打开终端(搜索里输入ter，回车)
  2. 新建自己的工作区(文件夹)，例如：(mkdir workspace)
  3. 进入到自己的工作区，例如：(cd workspace)
- 实际操作：
  1. 在自己的工作区内将远程github中的项目克隆到本地，例如：（git clone [url:这个是远程项目的网址]）ps:根据网速稳定性及项目大小可能克隆失败，
  2. 由于是初次克隆需要根据提示让输入github的登陆用户名和密码，输入回车即可.
  3. 在自己的工作区节点里查找自己的项目，例如：(输入 ls 回车 然后再 cd 命令进入项目)
  4. 在自己的项目节点查看项目是否完整，例如：(输入 ls 回车，对比github中的项目目录)

总结：完成以上步骤，Git项目管理入门操作准备工作已经完成

项目中常用的Git命令操作及流程

整体操作流程：



实际操作流程：

1.查看远程所有分支：git branch －r

在初期接触的时候还没有自己的本地分支，自己本身更改代码多的时候需要建立自己的本地分支，先把自己更改多的代码放在本地分支暂存起来，继续优先级高的任务。例如:(新建本地分支:xxx_temp ,git checkout -b xxx_temp; 查看本地分支：git branch ；切换到本地分支：git checkout xxx_temp)

2.切换到所需远程分支xxx：git checkout --track origin/xxx

切换本地分支，git checkout

3.更新本地代码：git pull --rebase

4.用xcode本地代码编译，检测是否编译成功，查错纠错沟通。

5.查看自己修改的代码： git status
如果不是自己确定要修改的代码，还原它（command +z）

6.修改的代码提交暂存区：git add .
这是提交所有修改后的代码到暂存区，根据需要也可以提交指定文件，例如:(git add [file name])

7.添加更改描述：git commit -m “我修改 / 添加了XXXXXX”
在添加描述的时候写得具体点

8.添加更改描述：git commit -m “我修改 / 添加了XXXXXX”

- 更新代码：git pull —rebase
- 编译本地代码
- 邀请Review
  - 由于是新手，在提交自己修改的代码的时候需要邀请所在团队熟悉项目的成员把一下关，防止产生新的问题，等到自己熟悉项目的时候，提倡邀请review
- 提交代码到远程服务器： git push XXX
  - push代码后再次更新一下代码，保证自己本地的代码是最新的

Git查看和逆向操作

查看

1.查看尚未暂存的更新：git diff

查看的结果为绿色为添加的内容红色为删除的内容

2.查看你commit的日志: git log

如果要查看指定文件的提交历史就是：git log -p <file>;以列表的形式查看指定文件的历史是：git blame <file>

逆向撤销

1.撤销目录中所有未提交的文件的修改内容：git reset --hard HEAD

2.撤销指定未提交的文件的修改内容: git checkout HEAD <file>

3.撤销指定的提交: git revert <commit>

用Git的时候需要注意什么?

1.修改和提交代码一定要仔细，因为每个人提交的代码都会影响到整个团队的开发。

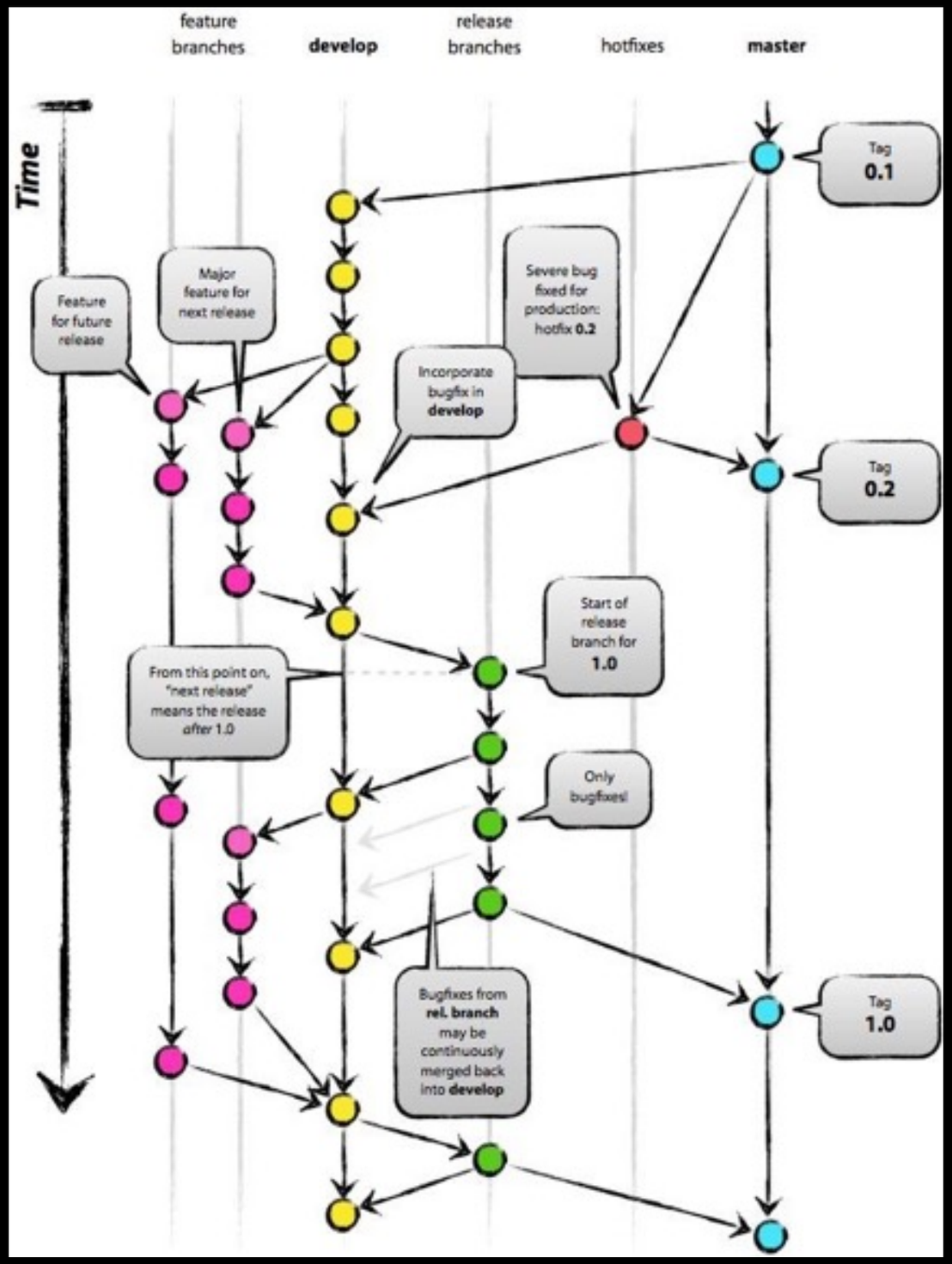
2.在每次用git add文件时，最好用git diff查看修改内容，确认这些内容是否需要修改? 是否有自己增加的无用的调试信息? 是否有不小心误删的内容? 代码对齐、整洁度是否良好等?

3.在每次准备提交代码时，必须先执行git pull命令，将远程服务器的代码同步到本地。然后尽量在较短的时间内完成git add、git comit、git push的动作。在执行git comit前再执行一次git pull，确认服务器端代码没有修改后，然后再执行完git comit，紧接着执行git push，将代码推送到服务器，避免中间有其他同事提交代码导致代码冲突而无法推送到服务器。一定要查看git push命令是否执行成功。

4.如有疑问请找同事帮忙，请勿在服务器上随意试验git命令。

Git难点

1.分支恐惧症



2.rebase恐惧症（git rebase用于把一个分支的修改合并到当前分支）

1> 我们知道，在某个分支上，我们可以通过git reset，实现将当前分支切换到本分支以前的任何一个版本状态，即所谓的"回溯"。即实现了本分支的"后悔药"。也即版本控制系统的初衷。

2> 还有另一种情况，当我们的项目有多个分支的时候。我们除了在本地开发的时候可能会"回溯"外，也常常会将和自己并行开发的别人的分支修改添加到自 己本地来。这种情况下很常见。作为项目管理员，肯定会不断的合并各个子项目的补丁，并将最新版本推送到公共版本库，而作为开发人员之一，提交自己的补丁之 后，往往需要将自己的工作更新到最新的版本库，也就是说把别的分支的工作包含进来。

3.conflict恐惧症

1.merge 和 patch（应用补丁）时产生冲突。

2.rebase就是重新设置基准，然后应用补丁的过程，所以也会冲突。

3.git pull会自动merge，所以git pull也会产生冲突

Git进阶学习

git官方文档：

https://help.github.com

25个 Git 进阶技巧：

http://www.open-open.com/lib/view/open1431331496857.html

git 冲突

http://my.oschina.net/u/1757458/blog/349827

Git速查表

Git 常用命令速查表

master :默认开发分支　　Head :默认开发分支
origin :默认远程版本库　　Head^ :Head 的父提交

创建版本库

```
$ git clone <url>          #克隆远程版本库
$ git init                 #初始化本地版本库
```

修改和提交

```
$ git status               #查看状态
$ git diff                 #查看变更内容
$ git add .                #跟踪所有改动过的文件
$ git add <file>           #跟踪指定的文件
$ git mv <old> <new>       #文件改名
$ git rm <file>            #删除文件
$ git rm --cached <file>   #停止跟踪文件但不删除
$ git commit -m "commit message"
                           #提交所有更新过的文件
$ git commit --amend       #修改最后一次提交
```

查看提交历史

```
$ git log                  #查看提交历史
$ git log -p <file>        #查看指定文件的提交历史
$ git blame <file>         #以列表方式查看指定文件
                           的提交历史
```

撤消

```
$ git reset --hard HEAD    #撤消工作目录中所有未提交
                           文件的修改内容
$ git checkout HEAD <file> #撤消指定的未提交文件的修
                           改内容
$ git revert <commit>      #撤消指定的提交
```

分支与标签

```
$ git branch               #显示所有本地分支
$ git checkout <branch/tag> #切换到指定分支或标签
$ git branch <new-branch>  #创建新分支
$ git branch -d <branch>   #删除本地分支
$ git tag                  #列出所有本地标签
$ git tag <tagname>        #基于最新提交创建标签
$ git tag -d <tagname>     #删除标签
```

合并与衍合

```
$ git merge <branch>       #合并指定分支到当前分支
$ git rebase <branch>      #衍合指定分支到当前分支
```

远程操作

```
$ git remote -v            #查看远程版本库信息
$ git remote show <remote> #查看指定远程版本库信息
$ git remote add <remote> <url>
                           #添加远程版本库
$ git fetch <remote>       #从远程库获取代码
$ git pull <remote> <branch> #下载代码及快速合并
$ git push <remote> <branch> #上传代码及快速合并
$ git push <remote> :<branch/tag-name>
                           #删除远程分支或标签
$ git push --tags          #上传所有标签
```

Git Cheat Sheet <CN> (Version 0.1)　　# 2012/10/26　-- by @riku　< riku@gitcafe.com / http://riku.wowubuntu.com >

Git命令手册

```
git init                                          # 初始化本地git仓库（创建新仓库）
git config --global user.name "xxx"               # 配置用户名
git config --global user.email "xxx@xxx.com"      # 配置邮件
git config --global color.ui true                 # git status等命令自动着色
git config --global color.status auto
git config --global color.diff auto
git config --global color.branch auto
git config --global color.interactive auto
git clone git+ssh://git@192.168.53.168/VT.git     # clone远程仓库
git status                                        # 查看当前版本状态（是否修改）
git add xyz                                       # 添加xyz文件至index
git add .                                         # 增加当前子目录下所有更改过的文件至index
git commit -m 'xxx'                               # 提交
git commit --amend -m 'xxx'                       # 合并上一次提交（用于反复修改）
git commit -am 'xxx'                              # 将add和commit合为一步
git rm xxx                                        # 删除index中的文件
git rm -r *                                       # 递归删除
git log                                           # 显示提交日志
git log -1                                        # 显示1行日志 -n为n行
git log -5
git log --stat                                    # 显示提交日志及相关变动文件
git log -p -m
git show dfb02e6e4f2f7b573337763e5c0013802e392818 # 显示某个提交的详细内容
git show dfb02                                    # 可只用commitid的前几位
git show HEAD                                     # 显示HEAD提交日志
git show HEAD^                                    # 显示HEAD的父（上一个版本）的提交日志 ^^为上两个版本 ^5为上5个版本
git tag                                           # 显示已存在的tag
git tag -a v2.0 -m 'xxx'                          # 增加v2.0的tag
git show v2.0                                     # 显示v2.0的日志及详细内容
git log v2.0                                      # 显示v2.0的日志
git diff                                          # 显示所有未添加至index的变更
git diff --cached                                 # 显示所有已添加index但还未commit的变更
git diff HEAD^                                    # 比较与上一个版本的差异
git diff HEAD -- ./lib                            # 比较与HEAD版本lib目录的差异
git diff origin/master..master                    # 比较远程分支master上有本地分支master上没有的
git diff origin/master..master --stat             # 只显示差异的文件，不显示具体内容
git remote add origin git+ssh://git@192.168.53.168/VT.git # 增加远程定义（用于push/pull/fetch）
```

```
git branch                                  # 显示本地分支
git branch --contains 50089                 # 显示包含提交50089的分支
git branch -a                               # 显示所有分支
git branch -r                               # 显示所有原创分支
git branch --merged                         # 显示所有已合并到当前分支的分支
git branch --no-merged                      # 显示所有未合并到当前分支的分支
git branch -m master master_copy            # 本地分支改名
git checkout -b master_copy                 # 从当前分支创建新分支master_copy并检出
git checkout -b master master_copy          # 上面的完整版
git checkout features/performance           # 检出已存在的features/performance分支
git checkout --track hotfixes/BJVEP933      # 检出远程分支hotfixes/BJVEP933并创建本地跟踪分支
git checkout v2.0                           # 检出版本v2.0
git checkout -b devel origin/develop        # 从远程分支develop创建新本地分支devel并检出
git checkout -- README                      # 检出head版本的README文件（可用于修改错误回退）
git merge origin/master                     # 合并远程master分支至当前分支
git cherry-pick ff44785404a8e               # 合并提交ff44785404a8e的修改
git push origin master                      # 将当前分支push到远程master分支
git push origin :hotfixes/BJVEP933          # 删除远程仓库的hotfixes/BJVEP933分支
git push --tags                             # 把所有tag推送到远程仓库
git fetch                                   # 获取所有远程分支（不更新本地分支，另需merge）
git fetch --prune                           # 获取所有原创分支并清除服务器上已删掉的分支
git pull origin master                      # 获取远程分支master并merge到当前分支
git mv README README2                       # 重命名文件README为README2
git reset --hard HEAD                       # 将当前版本重置为HEAD（通常用于merge失败回退）
git rebase
git branch -d hotfixes/BJVEP933             # 删除分支hotfixes/BJVEP933（本分支修改已合并到其他分支）
git branch -D hotfixes/BJVEP933             # 强制删除分支hotfixes/BJVEP933
git ls-files                                # 列出git index包含的文件
git show-branch                             # 图示当前分支历史
git show-branch --all                       # 图示所有分支历史
git whatchanged                             # 显示提交历史对应的文件修改
git revert dfb02e6e4f2f7b573337763e5c0013802e392818    # 撤销提交dfb02e6e4f2f7b573337763e5c0013802e392818
git ls-tree HEAD                            # 内部命令：显示某个git对象
git rev-parse v2.0                          # 内部命令：显示某个ref对于的SHA1 HASH
```

```
git reflog                                                  # 显示所有提交，包括孤立节点
git show HEAD@{5}
git show master@{yesterday}                                 # 显示master分支昨天的状态
git log --pretty=format:'%h %s' --graph                     # 图示提交日志
git show HEAD~3
git show -s --pretty=raw 2be7fcb476
git stash                                                   # 暂存当前修改，将所有至为HEAD状态
git stash list                                              # 查看所有暂存
git stash show -p stash@{0}                                 # 参考第一次暂存
git stash apply stash@{0}                                   # 应用第一次暂存
git grep "delete from"                                      # 文件中搜索文本“delete from”
git grep -e '#define' --and -e SORT_DIRENT
git gc
git fsck
```

THE END