索引

[1 git clone 3](#_Toc436230106)

[2 git status 3](#_Toc436230107)

[3 git add 4](#_Toc436230108)

[4 git commit 4](#_Toc436230109)

[5 git commit - -amend 4](#_Toc436230110)

[6 git rm 5](#_Toc436230111)

[7 git mv oldname newname 5](#_Toc436230112)

[8 git diff 5](#_Toc436230113)

[9 git diff - -cached 6](#_Toc436230114)

[10 git diff HEAD 6](#_Toc436230115)

[11 git checkout 6](#_Toc436230116)

[12 git rest HEAD 6](#_Toc436230117)

[13 git reset --soft HEAD^ 7](#_Toc436230118)

[14 git reset --mixed HEAD^ 7](#_Toc436230119)

[15 git reset --hard HEAD~1,2… 7](#_Toc436230120)

[16 git revert -n commit-ID 8](#_Toc436230121)

[17 git pull 8](#_Toc436230122)

[18 git push 8](#_Toc436230123)

[19 git log 9](#_Toc436230124)

[19.1 git log 输出信息介绍 9](#_Toc436230125)

[19.2 git log –p 9](#_Toc436230126)

[19.3 git log --stat 10](#_Toc436230127)

[19.4 git log --pretty=format 10](#_Toc436230128)

[19.5自定义格式 11](#_Toc436230129)

[19.6分枝拓扑图 12](#_Toc436230130)

[19.7利用提交查询过滤器可以查询经过某些条件限制后得到的提交信息 12](#_Toc436230131)

[19.7.1日期区间 12](#_Toc436230132)

[19.7.2提交者过滤器 13](#_Toc436230133)

[19.7.3可以指定完整人名或email地址来搜索 13](#_Toc436230134)

[19.7.4还可以通过邮箱”xxx@gionee.com”来搜索 14](#_Toc436230135)

[19.7.4查找提交信息 14](#_Toc436230136)

[19.7.5文件历史 14](#_Toc436230137)

[20 git branch 15](#_Toc436230138)

[21 git merge 16](#_Toc436230139)

[22 git rebase 17](#_Toc436230140)

[23 git cherry-pick 17](#_Toc436230141)

[24 git tag 17](#_Toc436230142)

[25 git filter-branch 18](#_Toc436230143)

[26 git stash 19](#_Toc436230144)

**Git 基本操作**

# 1 git clone

git clone 代码库的url

例如：git clone ssh://xxx@ip:29418/projectname

git clone -b test ssh://gerritroot@18.8.9.92:29418/test.git

克隆到本地的库，默认的代码库的当前分支(活动分支)名称是master，如果下载的时候，加上-b 分支名称，下载下来的代码库的当前分支（活动分支）就是指定的分支名称

git clone –b branch1 ssh://zhangjj@192.168.27.253:29418/test.git

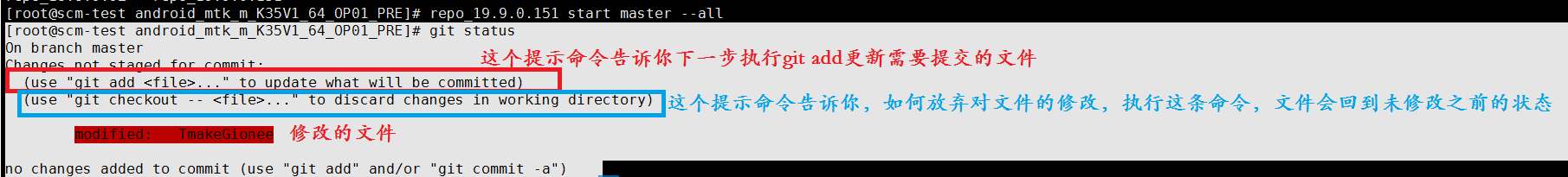
克隆到本地的库，默认的代码线是branch1

git clone 其他用法可以使用 git clone - -help查看

# 2 git status

查看当前版本库中文件的状态

如果对git的操作不熟练，在git库中做一次操作，可以使用git status查看版本库中文件的状态提示，提示可以帮助你向前走或向后退



# 3 git add

git add filename

git add directoryname

git add . 其中.表示当前目录下新添加的文件或目录，修改的文件或目录

作用：将工作区的修改提交到暂存区

# 4 git commit

git commit 与 git commit –m “message”相同，只是后者直接以-m参数将提交日志写入

git commit的作用是将暂存区的文件或目录提交到版本库区

# 5 git commit - -amend

git commit - -amend 的作用

1. 修正提交单中错误的用户名及邮箱

git commit - -amend --author=’zhangjj [zhangjj@gionee.com](mailto:zhangjj@gionee.com)’

2.代码走查不通过的时候，在本地使用这个命令可以不产生新的代码走查单，在本地修改完走查不通过的文件，需要执行git commit的时候，不要使用git commit –m，而是使用git commit --amend命令（前提本地代码库中的.git/hooks/commit-msg 钩子脚本是存在的，如果钩子脚本不存在，要到gerrit web页面的走查单中将changed-id xxxx贴到git commit --amend弹出的编辑框中）

# 6 git rm

git rm 文件名称

git rm –r 目录名称

将不需要的文件及目录从工作区及暂存区中删掉

1. 如果需要删除git库中已经受控的文件或目录请使用此命令，不要使用操作系统下的删除命令.受控-表示文件及目录状态为跟踪状态
2. 如果要删除第一次放到工作区的文件或目录，直接使用操作系统下的删除命令，也可以使用 git clean –df 移除不受控的文件或目录
3. git rm tt1.txt 将tt1.txt从工作区及暂存区删除，如果需要提交，下一步直接执行git commit –m “message”即可

# 7 git mv oldname newname

将代码库中受控的文件或者目录重命名

# 8 git diff

比对本地库中工作区与暂存区的差异

# 9 git diff - -cached

比对本地库中暂存区与库区的差异

# 10 git diff HEAD

比对本地库中工作区与库区的差异

# 11 git checkout

1. git checkout -- filename 或者git checkout -- directory

在本地git库的工作区修改或删除已受控的文件，后悔了，想要会退到之前的状态（工作区的文件回到修改之前的状态，暂存区，库区不受影响)

1. git checkout branchname 切换到某个分支
2. git checkout –b branch1 commit-ID 以某个提交修订版本为基准建立新分支，并切换到新分支

# 12 git rest HEAD

git rest HEAD filename

git reset HEAD 目录名

什么时候用：

执行了git add的动作之后,后悔了，想取消add的动作(工作区不受影响，库区不变，暂存区回到add之前的状态)

# 13 git reset --soft HEAD^

什么时候用：

在本地库中刚执行完添加及提交的动作，即执行git add ,git commit的动作之后，想取消git commit的动作，想重新commit一次（工作区，暂存区不变，库区回到commit之前的状态)

# 14 git reset --mixed HEAD^

什么时候用：

在本地库中刚完成一次提交，即执行git add ,git commit的动作之后，想取消add及commit的动作，但是保留工作区的修改：(暂存区，库区回到之前的状态)

# 15 git reset --hard HEAD~1,2…

如果你想彻底抛弃本地库中最新的提交，采用git reset --hard HEAD~1

如果你想抛弃本地库中最近两次的提交，采用git reset --hard HEAD~2

这种回退方式会将工作区、暂存区及版本库区最近1次或2次的修改全部抛弃

# 16 git revert -n commit-ID

* 修改或添加或删除了某个文件之后，执行git add ,git commit的动作之后,并且已经推送到远程库，如果回退远程库的此次合入呢？
* 在本地库通过git log找到此次提交的版本号(commit 40位哈希值)
* git revert -n commit-ID
* git commit –m “xxxxx”
* git push origin HEAD:refs/for/**remote\_branchname**

# 17 git pull

更新本地库

1. git pull --rebase

在本地库的A分支下执行该命令，表示将远程库中对应的A分支内容更新到本地A分支，加入参数 --rebase 可以避免在更新过程中本地库中产生合并节点

2.git pull origin branchname

将远程仓库的分支内容更新到本地库的当前分支上

# 18 git push

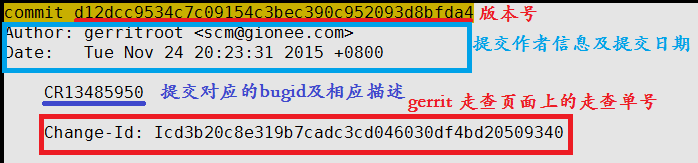
将提交到本地库中的内容推送到远程库中

git push origin HEAD:refs/for/branchname (推送到gerrit页面走查)

git push origin branchname (直接推送到远程git库中)

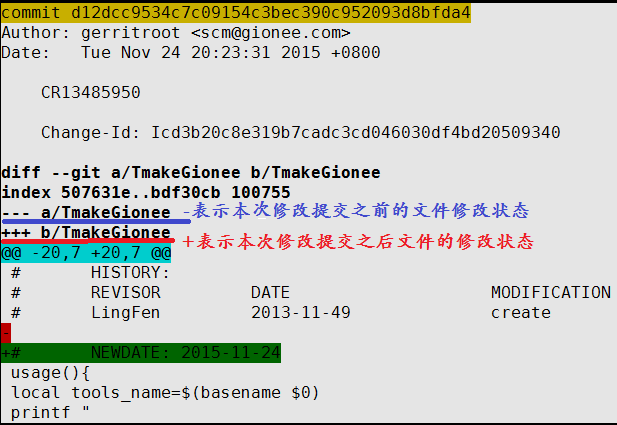
# 19 git log

## 19.1 git log 输出信息介绍

****

## 19.2 git log –p

用于查看每一次提交条目的补丁内容，这在代码复查时很管用，--表示本次提交版本之前文件的修改状态

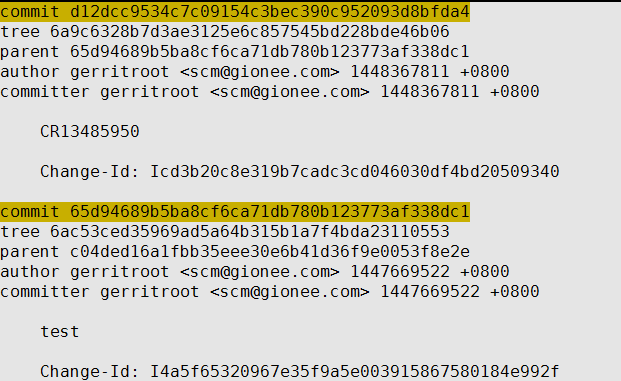


## 19.3 git log --stat

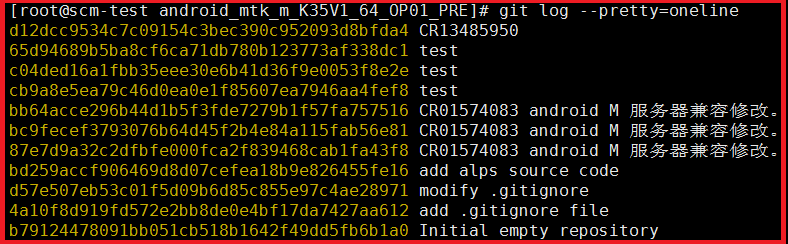
可以通过执行'git log -stat'命令，来显示被修改文件的修改统计信息，添加或删除了多少行

## 19.4 git log --pretty=format

git log --pretty=format'命令，可以将提交历史显示成你想要的格式。这里format的可选项包括：oneline，short，medium，full，fuller，email，raw。每种格式都有侧重的显示相关内容。

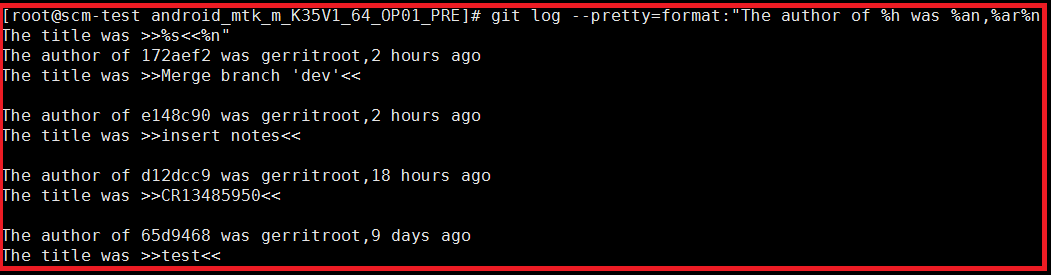


这里，最常用的一种格式是'oneline', 每一行表示一次提交，一行内容包括校验和与提交信息两项内容。这种显示格式可以很简洁的显示大量的提交信息



## 19.5自定义格式

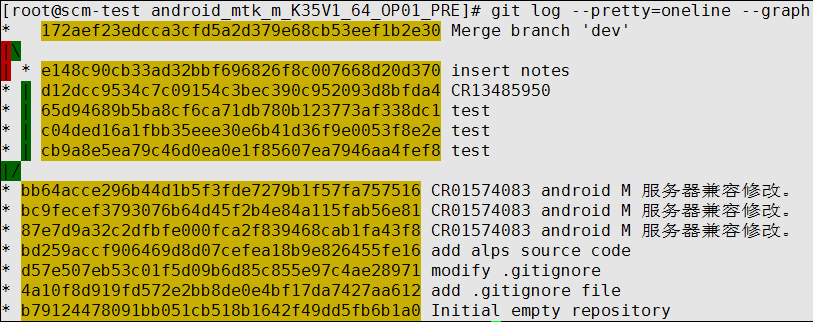
如果你希望git输出自定义的提交信息，你可以使用这样的命令：git log --pretty=format:"(format)".可以在""内自定义字符串与Git变量相结合的显示格式，在输出时，git会自动的将变量的位置输出为所需内容



更多详细自定义格式方法可以参考[**git log docs**](http://www.kernel.org/pub/software/scm/git/docs/git-log.html)

## 19.6分枝拓扑图

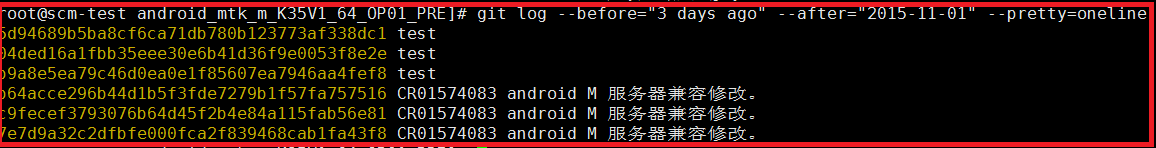
**还有一个日志输出选项是-graph，这个选项与--pretty=oneline结合不仅可以显示提交历史而且还会显示图形化的分枝拓扑**



## 19.7利用提交查询过滤器可以查询经过某些条件限制后得到的提交信息

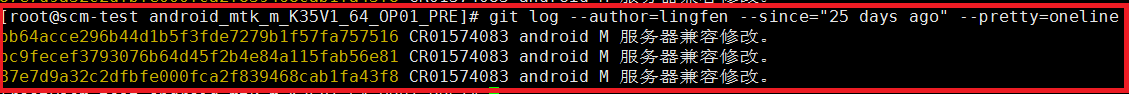
### ****19.7.1日期区间****

'**git log'命令后如果跟--before和--after选项，就会显示两个日期之间的提交条目。日期格式也有不同的可选格式。例如，我想查看2015-11-01之后，三天之前的修改，可以运行：**

****

### ****19.7.2提交者过滤器****

**大多数情况搜索的是作者，当然你可以通过--author或者--committer加名字的方式来搜索相应的提交条目。下面这个例子我们是执行命令来查找作者名（author）为lingfen，在过去25天内的所有提交条目：**

****

### ****19.7.3可以指定完整人名或email地址来搜索****

你也可以使用这些值的部分内容来查询，譬如，搜索Junio作者的提交内容，

[$ git log --author=Junio  
$ git log --author=Hamano  
$ git log --author="Junio C Hamano"  
$ git log --author=gitster@pobox.com](mailto:--author=gitster@pobox.com)

### ****19.7.4还可以通过邮箱”xxx@gionee.com”来搜索****



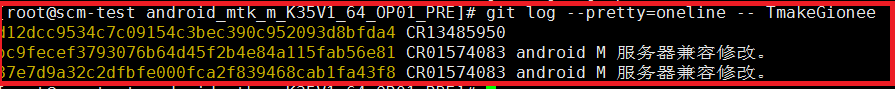
## ****19.7.4查找提交信息****

**如果你对“提交信息”更感兴趣，你可以通过提交信息里面的某个字符串来查找相应的提交。例如，下面是搜索在提交信息中含有'c90'的所有提交内容。**

$ git log --grep='C90' --pretty=oneline  
8c9e7947c2a76fb21bda11816c544d50e271156d http-push.c: squelch C90 warnings.

## ****19.7.5文件历史****

**还有的时候，你希望查看某个指定文件的提交历史。譬如你希望查看'notes.c'文件每一次的提交历史，你可以运行：**

****

**你还可以对文件目录进行这样的操作，譬如你想查看't/lib-httpd'目录的提交历史：**

$ git log --pretty=oneline --all -- t/lib-httpd/  
466ddf90c2f270b973d141f20e912f743743331c http-push: when making directories,   
603fb1168218a813f1b0816b1208c5d0c92cf29d Avoid apache complaining about lack   
faa4bc35a05ddb1822f3770cd8c51859e3b929ee http-push: add regression tests

**也可以加入多个查询项目，譬如要查看't/lib-httpd'目录或notes.c文件的提交历史：**

$ git log --pretty=oneline -- t/lib-httpd/ notes.c  
7b75b331f6744fbf953fe8913703378ef86a2189 Merge branch 'js/notes'  
466ddf90c2f270b973d141f20e912f743743331c http-push: when making directories,

# 20 git branch

1. git branch 在本地库执行git branch 可以查看本地库的分支，并且可以查看本地库当前分支(通常用\*标记)
2. git branch –r 在本地库中查看远程库的分支
3. git branch branname commit-ID

以某个修订版本建立新分支，commit-ID为40位的哈希值

1. git branch –d branchname

删除分支，再删除时会检查要删除的分支是否已经合并到其他分支，否则拒绝删除

1. git branch –D branchname

强制删除分支

1. git branch –m branch1 branch2

分支重命名，如果库中有同名的分支，则拒绝重命名

1. git branch –M branch1 branch2

强制重命名，不论库中是否有同名的分支

# 21 git merge

分支之间的合并，如果你想将分支dev的内容合并到主分支master，

git checkout master

git merge --no-ff dev

采用git merge的合并方式，代码线的演进会有合并节点产生，这样在查看主分支与开发分支的关系时会比较清晰明了

# 22 git rebase

1. git rebase通常称为变基，实现分支合并

在主分支中执行 git rebase dev 将dev分支的内容合并到主分支，代码线的演进将是一条直线，不会有节点，此时主分支与开发分支的历史关系就看不到了

1. git rebase -i 交换式应用可以改变提交历史顺序，改变日志，改变提交者**(这个过程会出现代码冲突，操作要慎重)**

# 23 git cherry-pick

分支拣选功能，例如现在dev分支上合入了几个修改点，但是只想将其中的一个修改点合入到主分支master，此时就要使用git cherry-pick

1. 切换到主分支

git checkout master

1. 在开发分支上找到需要部分功能的提交修定号通过git log找到(commit 40位的哈希值)
2. git cherry-pick commit-ID

# git tag

1. git tag –m “ xxxx” tagname commit-ID/HEAD/branchname

以某个分支最新的代码或者某个修订提交版本号创建一个带说明的标签

1. git tag –l

查看git库中的标签名称

1. git show tagname

显示tag内容

1. git checkout tagname

切换到标签中，查看对应的代码

# git filter-branch

1. 比如，你想在当前分支上**的所有**提交上山删除一个你不小心提交**的**密码文件。你可以执行**git** filter-branch --tree-filter 'rm -f password.txt' HEAD
2. 另比如，你想修改你的邮箱为新的邮箱，可如下：git filter-branch --commit-filter '

        if [ "$GIT\_AUTHOR\_EMAIL" = "oldemail@localhost" ];

        then

                GIT\_AUTHOR\_NAME="new name";

                GIT\_AUTHOR\_EMAIL="new email@example.com";

                git commit-tree "$@";

        else

                git commit-tree "$@";

        fi' HEAD

# git stash

**git** **stash** 可以用于暂存代码。 比如需要切换一个branch去做其他的事情，但是当前又有一些代码没有commit. 你显然也不会想要把这些修改放弃掉。

1.暂存当前代码。 可以暂存多个.依次压入堆栈

**git** **stash**

2.恢复暂存的代码

**git** **stash** pop

3.清除暂存代码

**git** **stash** clear