
git 常用命令及场景介绍

Revisions

Date	Version	Comments	Author
2014/4/1	V0.1	Create template	Xiaoyy2@lenovo.com
2014/5/9	V0.2	Complete the content	Xiaoyy2@lenovo.com
2014/5/27	V0.3	Update format, add content 4.2.5 , 5.9.6	Xiaoyy2@lenovo.com

GIT 常用命令及场景介绍	1
REVISIONS	1
1 名词解释:	3
2 参考文档	4
3 GIT/GERRIT 的配置:	4
3.1 GIT CONFIG	4
3.2 SSH CONFIG FOR GERRIT	4
4 代码获取与更新	6
4.1 第一次下载代码(GIT CLONE)	6
4.2 代码更新(GIT REMOTE/FETCH/PULL/RESET).....	6
5 代码的管理,查看,提交与清理	8
5.1 代码管理 --- Git 分支 (BRANCH)	8
5.2 查看代码历史 (GIT LOG)	9
5.3 查看代码改动 (GIT STATUS/DIFF)	10
5.4 清理代码(GIT CLEAN)	11
5.5 提交与撤销(GIT STATUS/ADD/RM/MV/COMMIT /RESET/REVERT)	11

5.6	更新修改或回退(GIT COMMIT --AMEND/ GIT REVERT).....	12
5.7	多次修改(GIT REBASE -I)	13
5.8	PATCH 相关操作	14
5.9	GITK 工具(GITK --ALL&)	15
5.10	GIT GUI 工具.....	18
6	代码推送 (GIT PUSH).....	20
6.1	PUSH 的规则: 什么情况下可以推送,什么情况下不能.....	20
6.2	如何推送代码	20
6.3	推送失败如何处理?	21
6.4	如何使用 GERRIT 网页查看推送结果	26
7	GIT 小技巧.....	28
7.1	GIT 命令自动补全 (GIT-COMPLETION.BASH).....	28
7.2	修改 GIT 命令行提示符颜色和格式 (GIT PS1)	28
7.3	GIT 别名 (GIT ALIAS)	28
7.4	更多小技巧	29

1 名词解释:

Init 初始化, 可以理解为新建一个 git, 执行 git init 后会在当前目录下生成 .git 目录, 有此目录即可执行 git 命令.

Clone 克隆, 从目标服务器得到整个用 Git 的拷贝。

Add 添加新的文件 (文件夹) 到 Git 项目中, 如果添加文件夹, 该文件夹下所有文件将被包含。

同时可以使用 rm 和 mv 从 git 项目中删除和重命名文件 (文件夹)。

Commit 告诉 Git 你想要记录现在的操作, Git 会保留一个当前修改后的文件的快照。

Push 把当前的这份拷贝提交到代码服务器(或 gerrit 服务器)

Pull 等于 fetch 加上 merge, 如果你很久前 clone 得到某项目的一份拷贝, 用 pull 可以更新到最新版本。

Fetch 得到远端仓库分支的 head ref, 找出本地对象数据库所缺少的对象, 并把它们下载下来。

Reset 如果你正在编辑的文件乱了, 可以选择从重新开始编辑, 通常是选择恢复到上一个编辑点。

Checkout 可以理解为在 branch 间切换, 也可以切换代码到指定的提交点。

Branch 可以理解理解为两个子版本, 当前版本出现了两个不同分支

Master git 创建时默认的分支名

Merge 代码的合并, 如果有冲突需要手动处理。

Cherry-pick 意味着从一系列的修改中选出一部分修改(通常是提交), 只把 patch 应用到当前代码中

Diff 可以对每一步操作的结果进行对比, 如 add 前后, commit 前后, commit 之前的比较。

Revert 可以回退指定的 commit 内容

Repository 可以理解为 git 的数据库, 修改并 commit 后, 一个文件快照被推送到这里, 被保存起来

Working tree 可以理解为未经任何修改刚刚 clone 下来的工作目录。

Index (cache/staging area) git 的缓存, 包括所有修改但是还没有 commit 的文件, 记录了当前修改状态的 working tree。

HEAD 可以理解为一个指针变量, 永远指向当前所 checkout 的点, 在 gitk 中黄颜色的点就代表了 HEAD 所在。

HEAD~n 代表往前数第 n 个祖先节点, HEAD~=HEAD^, HEAD~2=HEAD^^, HEAD~3=HEAD^^^ ...

HEAD^n 代表当前点的第几个父提交(多分支 merge 才会有多个父提交)

SHA1 由 40 位 16 进制数字组成的字符串, 是 git 存储对象的哈希值, 可以唯一标识某个对象,

如 commit ID 即唯一代表了这个 commit 对象, 通常取前 6 位数字即可唯一代表。

Remote 服务器端的路径, 常用别名标识, 在分支中可见 remotes/origin/test_br1 中 origin 即 remote 的别名

可用 git remote -v 命令查看

Origin 默认的 remote 名字, 如果用 git clone 命令下载的代码, remote 都是 origin, 可以用 git remote 命令查看。

Upstream 就是指服务器端, downstream 指 local 端

Ref 指 git 存储的分支, tag 等对象, 用 40 字节的 SHA1 来唯一标识, 可在 .git/refs/ 目录下查看

Refspec 一种服务器上和本地的 ref 之间的映射关系的说明(.git/config), 在 pull, push 等操作时进行关联

Parent/Child

从 commit 的提交历史来看, 同一条线上先提交的 commit 就是后提交 commit 的 parent, 反之为 child

Merge 后的那个节点会有多个 parents, 分支分开的那个节点会有多个 childs

Rebase To reapply a series of changes from a branch to a different base, and reset the head of that branch to the result.

fast-forward

是 merge 的一种特殊但是又很常见的形式, 当 merge 的对象刚好是你本地的子节点, git 会自动更新到这个节点而不再生成一个新的 merge commit. 本地的 commit 推送到服务器后再执行 fetch 更新就是 fast-forward

Change-ID

is the unique identity assigned to this change. It does not match the commit id, To avoid confusion with commit names,

Change-Ids are typically prefixed with an uppercase I.

更多术语 : http://gitbook.liuhui998.com/7_8.html

2 参考文档

Pro Git 中文版 <http://git-scm.com/book/zh/>

3 Git/Gerrit 的配置:

3.1 Git config

git 最常用的是配置存放在 ~/.gitconfig, 每个人都需要自己进行设置后才能下载,提交代码.

对应的配置命令要加 --global 选项。如:

```
git config --global user.name [username]
git config --global user.email [email]
git config --global --replace-all url.sma:.pushInsteadOf smamirr:
```

也可以直接编辑 ~/.gitconfig 文件填写内容, 可参考我的配置如下:

```
[user]
  name = xiaoyy2
  email = xiaoyy2@lenovo.com
[core]
  autocrlf = true
  editor = vim
[url "sma:"]
  pushInsteadOf = smamirr:
[alias]
  ci = commit -a -v
  co = checkout
  st = status
  br = branch
  throw = reset --hard HEAD
  throwh = reset --hard HEAD^
[color]
  ui = true
[push]
  default = current
[merge]
  tool = bc3
```

系统级的配置文件/etc/gitconfig, 一般用户无须修改. 对应的配置命令要加 --system 选项。

```
git config --system receive.fsckObjects true
```

Git 会在每次推送生效前检查库的完整性, 确保有问题的客户端没有引入破坏性的数据。

代码库级别的 .git/config , 一般用户无须修改. 对应的配置命令要加 --local 选项。如:

```
git config --local core.filemode true
```

git 就会对文件的 filemode 更改也会显示出来。

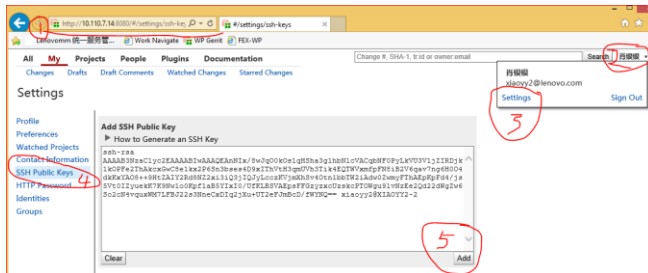
3.2 ssh config for Gerrit

目的是为了让两个 linux 机器之间使用 ssh 不需要用户名和密码, 采用了数字签名 RSA 来完成这个操作:

ssh-keygen -q -t rsa -f ~/.ssh/id_rsa -P "", 将会在~/.ssh/目录下生成密钥文件和私钥文件 id_rsa,id_rsa.pub

将 id_rsa.pub 文件复制到服务器的 .ssh 目录, 并 cat id_dsa.pub >> ~/.ssh/authorized_keys 即可.

在 Gerrit 中的配置如下图:



平台 team 提供的 dinit 脚本已经可以自动完成 git config 的配置和 ssh key 的生成, 过程中我们只需要与脚本交互输入姓名和邮箱。

```
[xiaozy2@XIAOZY2-2] /usr$
$cd
[xiaozy2@XIAOZY2-2] ~$
$init
notice that if what in [] is same as what you want,press ENTER directly
pls input your git user name[xiaozy2]:
pls input your git user email[xiaozy2@lenovo.com]:
Enter file in which to save the key [/usr/.ssh/id_rsa]:
pls goto your gerrit site: http://sc.lenovo.com:8080
and copy rsa public id to gerrit setting:

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQEAhN1x/8wJqO0k0c1qHSha3g1hbN1cUACqbNF8PyLkUu3U1jZIR
09xIThUth3qnlUth3Tik4EQTUWxmfpFN81B2U6qau7ng6H004dkKcYA08++9HtZa1Y2Bd8NZ2xi3iQ3j1QJvLc
KqFq44jse5UteB1ZyuekK7K9Nwlo0Kpf1aB5V1xi0/UfKLBSUAEpsFFGzgzxcUzskePTOMgu91oNcEe2Qd22dUy
+UT2eFJmBcD/fUWQ== xiaozy2@XIAOZY2-2

pls make sure you have add the public key to gerrit< http://sc.lenovo.com:8080 >!
```

同时会生成一个~/.ssh/config 文件, 内容如下:

```
Host sma
  Hostname sc.lenovo.com
  Port 29418
  User xiaozy2

Host wp
  Hostname 10.110.7.14
  Port 29418
  User xiaozy2
```

这个配置可以用别名代替服务器,端口,用户名, 简化代码下载的命令, 如:

下载代码的常规命令是:

```
git clone -u xiaozy2@sc.lenovo.com:29418/platform/manifest.git -b master
```

有了这个配置后就可以简化为: git clone sma:platform/manifest.git -b master

测试配置是否正确, 服务器链接是否正常, 如果显示 Welcome to Gerrit Code Review 就说明没问题了.

```
[xiaozy2@XIAOZY2-2] ~$
$ssh up
The authenticity of host '[10.110.7.14]:29418 ([10.110.7.14]:29418)' can't be established.
RSA key fingerprint is f9:11:6c:eb:10:de:ea:89:1c:23:96:e5:61:1e:e3:85.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[10.110.7.14]:29418' (RSA) to the list of known hosts.

**** Welcome to Gerrit Code Review ****

Hi 肖银银, you have successfully connected over SSH.

Unfortunately, interactive shells are disabled.
To clone a hosted Git repository, use:

git clone ssh://xiaozy2@10.110.7.14:29418/REPOSITORY_NAME.git

Connection to 10.110.7.14 closed.
```

4 代码获取与更新

4.1 第一次下载代码(git clone)

克隆仓库的命令格式为 `git clone [url]`, [url]为你想要复制的项目,后跟-b 分支名, 如果不带-b 参数默认为 master 分支, 比如:

```
git clone ssh://xiaoyy2@10.110.7.14:29418/platform/8926WP -b pioneer_dev
```

此命令可在 [gerrit 网页上的 Projects→List→platform/8926WP→General→clone→SSH](#) 中查看(见图 1.1.1)

其中 url 部分为:

`ssh://`表示使用的是 ssh 协议

`xiaoyy2@10.110.7.14:29418` 表示 `用户名@服务器地址:端口号`, 服务器地址可以是 IP 或机器名,端口号固定为 29418

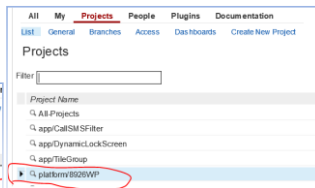
`platform/8926WP` 是 git 在服务器上的路径, 可在 [gerrit 网页上的 Projects→List](#) 中查看(见图 1.1.2)

-b 后跟分支名, 如果不带-b 参数, 表示取默认分支 master

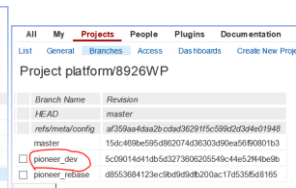
`pioneer_dev` 是分支名, 可在 [gerrit 网页上的 Projects→List→platform/8926WP→Branches→Branch Name](#) 中查看(见图 1.1.3)



(图 1.1.1)



(图 1.1.2)



(图 1.1.3)

若`~/ssh/config`中配置了代码服务器信息(见 [ssh config for Gerrit](#)), git clone 命令可简化为:

```
git clone wp:/platform/8926WP -b pioneer_dev
```

4.2 代码更新(git remote/fetch/pull/reset)

简言之就是用 `git remote` 管理你的远程仓库, 用 `git fetch/pull` 更新你的项目, 用 `git reset` 在本地代码分支间进行切换。

4.2.1 Git remote

Remote 信息描述了本地 git 与服务器端 git 的一种映射关系。

不带参数执行 `git remote` 会返回远程仓库的别名, 带-v 参数还可以看到每个别名的实际链接地址, 如:

```
$ git remote
origin
$ git remote -v
origin wp:/platform/8926WP (fetch)
origin wp:/platform/8926WP (push)
```

这里的 origin 是默认的远程仓库名字, 一般用 `git clone` 命令下载的代码默认远程仓库名字都是 origin

同一个 git 仓库可以有多个 remote, 在 `git branch -r` 中会看到不同 remote 的分支名前缀也不同
如 `remotes/wp/master` 和 `remotes/sma/master` 就分别代表了 2 个不同远端仓库的 master 分支.
增加远程仓库 `git remote add [alias] [url]`, 此命令将 `[url]` 以 `[alias]` 为别名添加为本地的远端仓库。
删除远程仓库 `git remot rm [alias] [url]`, 此命令将 `[url]` 以 `[alias]` 为别名的远端仓库从本地删除。

```
$ git remote -v
origin  wp:platform/8926WP (fetch)
origin  wp:platform/8926WP (push)

$ git remote add github git@github.com:schacon/hw.git

$ git remote -v
origin  wp:platform/8926WP (fetch)
origin  wp:platform/8926WP (push)
github  git@github.com:schacon/hw.git (fetch)
github  git@github.com:schacon/hw.git (push)

$ git remote rm github git@github.com:schacon/hw.git

$ git remote -v
origin  wp:platform/8926WP (fetch)
origin  wp:platform/8926WP (push)
```

4.2.2 Git fetch

Fetch 命令只从远端仓库下载新分支与数据, 即只将服务器的数据更新到本地的数据库中, 并不会 checkout, 也不会对本地的工作造成任何修改.

<code>git fetch origin</code>	只更新别名为 origin 的远端仓库数据到本地.
<code>git fetch --all</code>	会更新所有的远端仓库数据到本地.

4.2.3 Git pull

基本上, 该命令就是在 `git fetch` 之后紧接着 `git merge` 远端分支到你所在的任意分支.

比如你当前 checkout 的本地分支是 `test_br1`

如果执行 `git pull origin master` 就会将远端 origin 的 master 分支更新到本地, 并与 `test_br1` 进行合并

如果执行 `git pull --rebase origin master` 将会将本地的 `test_br1` 分支 rebase 到 master 分支上

4.2.4 Git reset

用 hard reset 可以 checkout 到任意指定的分支/提交点.

<code>git reset --hard</code>	会把 <code>git status</code> 中能看到的 所有 改动都删除。
<code>git reset db9adcf --hard</code>	会直接 checkout 到 db9adcf 这个 commit.

4.2.5 Git checkout 更新某个文件(或目录)到指定版本

Git checkout **db9adcf** file1 将会只取 **db9adcf** 这个 commit 的 file1 这个文件的内容, 其余内容不变

Git checkout **db9adcf** folder1 将会只签出 **db9adcf** 这个 commit 中 folder1 这个目录的内容, 其余内容不变

5 代码的管理,查看,提交与清理

5.1 代码管理 --- Git 分支 (branch)

5.1.1 什么是分支

分支其实就是从某个提交对象往回看的历史，本质上仅仅是个指向 commit 对象的可变指针。Git 会使用 master 作为分支的默认名字。在若干次提交后，你其实已经有了一个指向最后一次提交对象的 master 分支，它在每次提交的时候都会自动向前移动。

不在分支上的 commit 也叫做 detached 对象，没有分支或 tag 进行管理，在 gitk 界面刷新后就不会再显示，而且会被 git 后台自动清理，因此我们在修改代码前一定要先创建一个分支，checkout 这个分支后再开始工作。

5.1.2 创建，查看，切换分支 (git branch)

git branch 列出可用的本地分支

没有参数时，git branch 会列出你在本地的分支。你所在的分支的行首会有个星号作标记。

git branch <branchname> 创建新分支

git branch -r 列出 remote 分支

git branch -a 列出所有本地和 remote 的分支

git checkout <branchname> 切换分支

git checkout -b <branchname> 创建新分支并切换到该分支

```
[xiaoyu2@X1A0VY2-21 /c/workspace/F1/tools]$ git branch
* person1
[xiaoyu2@X1A0VY2-21 /c/workspace/F1/tools]$ git branch master
[xiaoyu2@X1A0VY2-21 /c/workspace/F1/tools]$ git branch
* master
* person1
[xiaoyu2@X1A0VY2-21 /c/workspace/F1/tools]$ git branch -r
origin/HEAD -> origin/master
origin/master
origin/phone/wp_test
origin/pioneer_dev
origin/pioneer_rebase
[xiaoyu2@X1A0VY2-21 /c/workspace/F1/tools]$ git branch -a
* master
remotes/origin/HEAD -> origin/master
remotes/origin/master
remotes/origin/phone/wp_test
remotes/origin/pioneer_dev
remotes/origin/pioneer_rebase
[xiaoyu2@X1A0VY2-21 /c/workspace/F1/tools]$ git checkout master
Switched to branch 'master'
[xiaoyu2@X1A0VY2-21 /c/workspace/F1/tools]$ git branch
* master
* person1
[xiaoyu2@X1A0VY2-21 /c/workspace/F1/tools]$ git checkout -b newbranch
Switched to a new branch 'newbranch'
[xiaoyu2@X1A0VY2-21 /c/workspace/F1/tools]$ git branch
* master
* newbranch
* person1
[xiaoyu2@X1A0VY2-21 /c/workspace/F1/tools]$
```


5.1.3 分支改名,删除(git branch -m/-d)

分支改名: Git branch -m <current_branch_name> <new branch name>

删除分支: git branch -d <branch name> / git branch -D <branch name>

不能删除当前 checkout 的分支

```
[xiaoyy2@XIAOYV2-21 /c/workspace/P1/tools]$ git branch
* master
* newbranch
  person1
[xiaoyy2@XIAOYV2-21 /c/workspace/P1/tools]$ git branch -m newbranch new_br1
[xiaoyy2@XIAOYV2-21 /c/workspace/P1/tools]$ git branch
* master
* new_br1
  person1
[xiaoyy2@XIAOYV2-21 /c/workspace/P1/tools]$ git branch -d person1
Deleted branch person1 (was 56ff7b6a).
[xiaoyy2@XIAOYV2-21 /c/workspace/P1/tools]$ git branch
* master
* new_br1
[xiaoyy2@XIAOYV2-21 /c/workspace/P1/tools]$ git branch -D new_br1
error: Cannot delete the branch 'new_br1' which you are currently on.
[xiaoyy2@XIAOYV2-21 /c/workspace/P1/tools]$ git branch -d new_br1
error: Cannot delete the branch 'new_br1' which you are currently on.
[xiaoyy2@XIAOYV2-21 /c/workspace/P1/tools]$
```

5.2 查看代码历史 (git log)

Git log 查看从当前 HEAD 开始本分支所有的提交历史

```
[xiaoyy2@XIAOYV2-21 /c/workspace/P1/tools]$ git log
commit 982536da3ced6d8f74ba9ae192aa73a769af22e
Author: xiaoyy2 <xiaoyy2@lenovo.com>
Date:   Mon May 5 13:14:00 2014 +0800

    xyy: add test commit 2

Change-Id: Ibf404703788fdac71985c84caba387ech20ab81e
commit d95c1e5e752488fac3c1728875a22b46703b8b92
Author: xiaoyy2 <xiaoyy2@lenovo.com>
Date:   Mon May 5 12:57:16 2014 +0800

    xiaoyy2: test commit 1

Change-Id: I73123b0dec3a8426e4089b4440de63d4a637aa5
commit fa0241c97af7277d3113ed46ccaf6d324c08e64
Author: xiaoyy2 <xiaoyy2@lenovo.com>
Date:   Tue Apr 29 17:42:02 2014 +0800
```

Git log -1 查看一个提交信息

```
[xiaoyy2@XIAOYV2-21 /c/workspace/P1/tools]$ git log -1
commit 982536da3ced6d8f74ba9ae192aa73a769af22e
Author: xiaoyy2 <xiaoyy2@lenovo.com>
Date:   Mon May 5 13:14:00 2014 +0800

    xyy: add test commit 2

Change-Id: Ibf404703788fdac71985c84caba387ech20ab81e
[xiaoyy2@XIAOYV2-21 /c/workspace/P1/tools]$
```

Git log -1 -p 查看一个提交的内容

```
[xiaoyy2@XIAOYV2-21 /c/workspace/P1/tools]$ git log -1 -p
commit 982536da3ced6d8f74ba9ae192aa73a769af22e
Author: xiaoyy2 <xiaoyy2@lenovo.com>
Date:   Mon May 5 13:14:00 2014 +0800

    xyy: add test commit 2

Change-Id: Ibf404703788fdac71985c84caba387ech20ab81e
diff --git a/build8926.bat b/build8926.bat
index 80f9e72..c20e16b 100644
--- a/build8926.bat
+++ b/build8926.bat
@@ -1,4 +1,4 @@
-@ECHO OFF
+@ECHO OFF
+
+IF /I "%1"=="help" ( GOTO HELP ) else ( GOTO SET_ENV )
diff --git a/testadd.txt b/testadd.txt
index 80f30f4..137e2ba 100644
--- a/testadd.txt
+++ b/testadd.txt
@@ -2,4 +1,4 @@
 test add line1
+
+No newline at end of file
 test add line2
+
+No newline at end of file
[xiaoyy2@XIAOYV2-21 /c/workspace/P1/tools]$
```

Git log -5 --oneline 简洁显示 5 个提交历史信息

```

[xiaoyy2@XIAOY2-21 /c/workspace/ROOT_GIT]
$git log -5 --oneline
982536d xyy: add test commit 2
d95c1e5 xiaoyy2: test commit 1
fa02416 [Pioneer][Camera][weicla] update minorSV to 0050
5268ad3 Commit label r10610050.1 <wapi>
7e8a13e Commit label r10610050.1
[xiaoyy2@XIAOY2-21 /c/workspace/ROOT_GIT]
$

```

git log --graph --oneline --all --decorate

```

[xiaoyy2@XIAOY2-21 /c/workspace/ROOT_GIT]
$git log --graph --oneline --all --decorate
* 982536d (HEAD, pioneer_dev) xyy: add test commit 2
* d95c1e5 xiaoyy2: test commit 1
  * d22ad1 [origin/pioneer_dev] rebase Commit label r10610053.1 <wapi>
    * 0654da0 Commit label r10610053.1
  * fa02416 [origin/pioneer_dev] [Pioneer][Camera][weicla] update minorSV to 0050
  * 5268ad3 Commit label r10610050.1 <wapi>
  * 7e8a13e Commit label r10610050.1
  * c9994be Commit label r10610046.3 <wapi>
  * c58ba22 Commit label r10610046.3
  * 8187806 Commit label r10610042.2 <wapi>
  * 110c97e Commit label r10610042.2

```

git log d95c1e5...5268ad3 --oneline 可以查看 2 个 commit 直接的提交记录信息

```

[xiaoyy2@XIAOY2-21 /c/workspace/ROOT_GIT]
$git log d95c1e5...5268ad3 --oneline
d95c1e5 xiaoyy2: test commit 1
fa02416 [Pioneer][Camera][weicla] update
[xiaoyy2@XIAOY2-21 /c/workspace/ROOT_GIT]
$

```

5.3 查看代码改动 (git status/diff)

Git status 显示所有文件改动信息，比如

```

[xiaoyy2@XIAOY2-21 /c/workspace/ROOT_GIT]
$git status
On branch pioneer_dev
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   build8926.bat
    renamed:    contentx.txt -> contentx_new.txt

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   build8926.bat
    deleted:    testadd.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    newfile.txt
    testadd.txt

```

Git diff 显示所有"not staged"的文件改动内容，也就是 git status 显示红色的文件修改部分，不包括新增文件

```

[xiaoyy2@XIAOY2-21 /c/workspace/ROOT_GIT]
$git diff
diff --git a/build8926.bat b/build8926.bat
index e69ea77..c369e84 100644
--- a/build8926.bat
+++ b/build8926.bat
@@ -1,5 +1,5 @@
ECHO OFF

add new test line
IF /I "%1"=="help" < GOTO HELP > else < GOTO
:LOOP
diff --git a/testadd.txt b/testadd.txt
deleted file mode 100644
index 134e2ba..0000000
--- a/testadd.txt
+++ /dev/null
@@ -1,3 +0,0 @@
test add line1
test add line2
test add line3
\ No newline at end of file
[xiaoyy2@XIAOY2-21 /c/workspace/ROOT_GIT]
$

```

Git diff --cached 显示"staged"状态的文件改动内容，也就是 git status 显示绿色的文件修改部分

```
[xiaoyy2@XIAOYY2-21 /c/workspace/ROOT_GIT]
$git diff --cached
diff --git a/build8926.bat b/build8926.bat
index c70a16b..e59ea77 100644
--- a/build8926.bat
+++ b/build8926.bat
@@ -1,4 +1,4 @@
@ECHO OFF
@ECHO OFF

IF /I "%1"=="help" < GOTO HELP > else <
diff --git a/contents.xml b/contents.xml
deleted file mode 100755
index bd4ce5a..0000000
--- a/contents.xml
+++ /dev/null
@@ -1,45 +0,0 @@
<?xml version="1.0" ?>
<!--
=====
contents.xml
```

git diff <SHA1> <SHA1> 查看 2 个 commit 之间的不同,本例中第一个 SHA1 为 dd4cdaa, 第二个 SHA1 为 251ec29, 可以任意指定

```
[xiaoyy2@XIAOYY2-21 /c/workspace/ROOT_GIT]
$git diff dd4cdaa 251ec29
diff --git a/build8926.bat b/build8926.bat
index 57d970f..21afd3f 100644
--- a/build8926.bat
+++ b/build8926.bat
@@ -615,12 +615,8 @@ SETLOCAL EnableDelayedExpansion
set Time_Hm2=%time:"1.1%%time:"3.2
set Time_Random=%Time_Hm2.%time:"6
Eren for PhoneFirmwareRevision
if not defined majorSU <
- set majorSU=1061
-
- if not defined minorSU <
- set minorSU=0037
-
+ set majorSU=1061
+ set minorSU=0037
set majorOEM=7788
set tsp="%time:"0.1%"
if %tsp%==" " <
```

5.4 清理代码(git clean)

记住一个 git clean -fdx 命令就行了

5.5 提交与撤销(git status/add/rm/mv/commit /reset/revert)

在 checkout 本地分支 pioneer_dev 分支后,我新增了 newfile.txt, 删除了 testadd.txt 文件, build8926.bat 修改了 2 次, 第一次修改(见 staged area)后我执行了 git add build8926.bat, 第二次修改(见 unstaged area) 没有执行 git add 命令. 还有一个文件重命名,命令是 git mv contents.xml contents_new.xml
Git status 结果如下:

```
[xiaoyy2@XIAOYY2-21 /c/workspace/ROOT_GIT]
$git status
# On branch pioneer_dev
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   build8926.bat
#       renamed:    contents.xml -> contents_new.xml
#
+-----+
+ Changes not staged for commit:
+   (use "git add/rm <file>..." to update what will be committed)
+   (use "git checkout -- <file>..." to discard changes in working directory)
+
+       modified:   build8926.bat
+       deleted:    testadd.txt
+
+-----+
+ Untracked files:
+   (use "git add <file>..." to include in what will be committed)
+
+       newfile.txt
[xiaoyy2@XIAOYY2-21 /c/workspace/ROOT_GIT]
$
```

Unstaged

Staged

Comitted

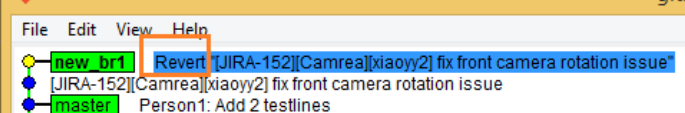
根据提示信息我们可以看到文件被划分为 3 种情况, 1 为绿色的 staged 部分, 2 为红色的 not staged 部分 (其中包含 untracked 新增文件), 以下是这 3 种



本地提交代码(commit)时需要带上 commit message, 一般项目会对这个格式是有一定要求, 比如用英文说明改动摘要, 包含 bug 号, 作者名字, 修改的模块等, 如: `git commit -m "[JIRA-152][Camera] [xiaoyy2] Fix front camera rotation issue"`
Add 和 commit 可以合并到一步进行: `git commit -am "[JIRA-152]..."`

需要回退提交可以用 revert 命令: `git revert <SHA1>`

```
xiaoyy2@XIAOYY2-21 /c:/workspace/P1/tools
$ git commit -am "[JIRA-152][Camreal][xiaoyy2] fix front camera rotation issue"
[new br] 1bc516c [JIRA-152][Camreal][xiaoyy2] fix front camera rotation issue
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 a
[xiaoyy2@XIAOYY2-21 /c:/workspace/P1/tools
$ git log -1 --oneline
1bc516c [JIRA-152][Camreal][xiaoyy2] fix front camera rotation issue
[xiaoyy2@XIAOYY2-21 /c:/workspace/P1/tools
$ git revert 1bc516c
[new br] e02dc8e1 Revert "[JIRA-152][Camreal][xiaoyy2] fix front camera rotation iss
1 file changed, 0 insertions(+), 0 deletions(-)
delete mode 100644 a
[xiaoyy2@XIAOYY2-21 /c
$ gitk --all&
[1] 6404
[xiaoyy2@XIAOYY2-21 /c
$
```

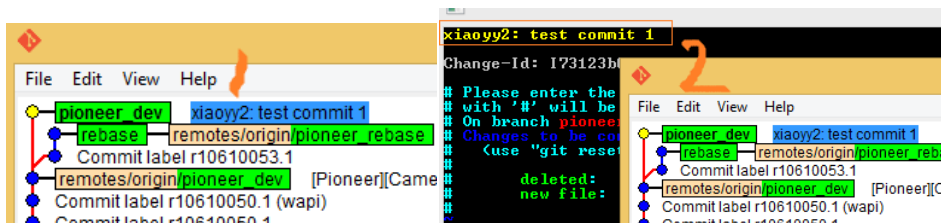


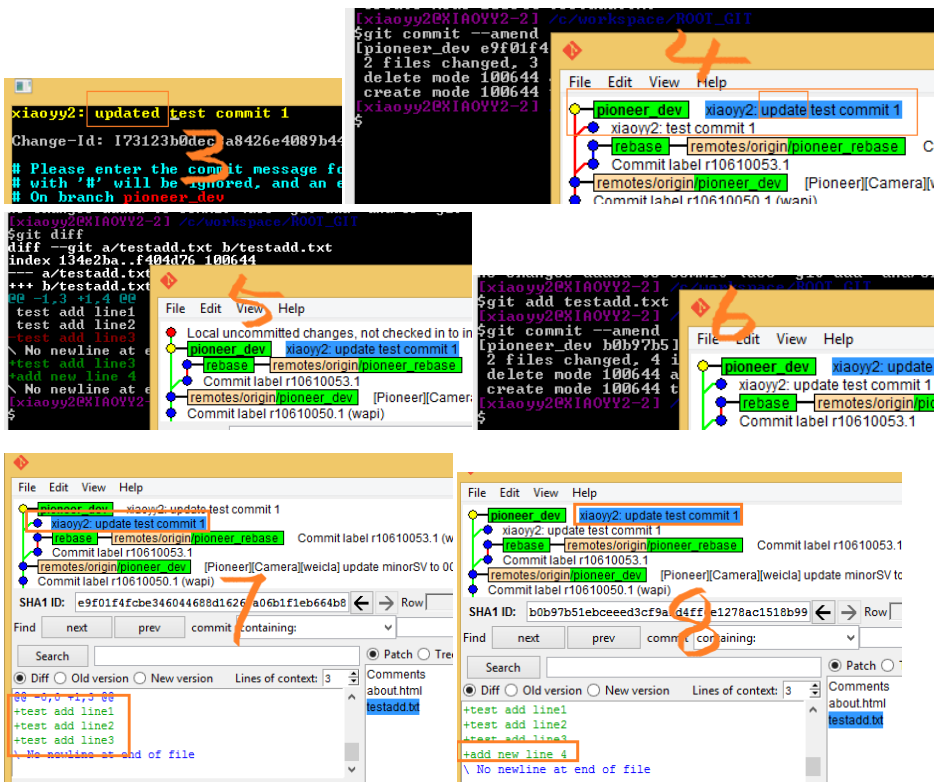
5.6 更新修改或回退(git commit --amend/ git revert)

批注 [HWW1]: 修订上个版本

场景 1: 对图 (1) 中已经 commit 过的 commit message 进行修改, 只要在命令行窗口执行 `git commit --amend`, 然后在弹出的图 (2) 中编辑, 如图 (3) 中我增加了一个 update 字, 然后:wq 退出后在 gitk 界面 F5 刷新即可看到图 (4) 中更新的结果。

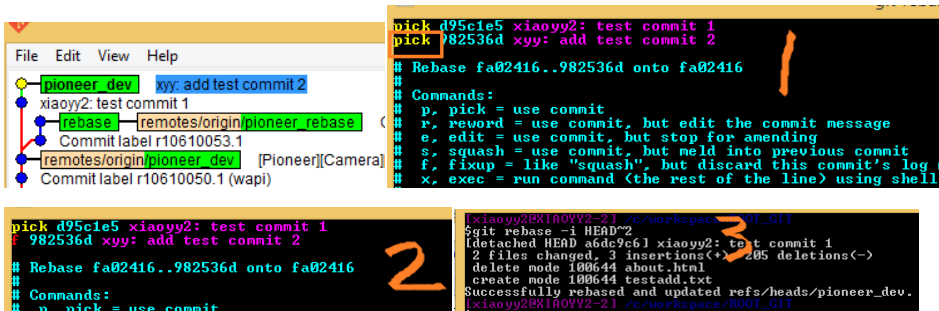
场景 2: 对图 (4) 中已经 commit 过的文件内容进行修改并再次提交, 如图 (5) 对文件更新后执行 `git add` (图 6), 然后在 commit 的时候直接选择 `git commit --amend`, 在编辑 commit message 界面直接:wq 保存退出, 然后在 gitk 界面 F5 刷新查看结果 (图 5, 图 6), 可以看到 testadd.txt 文件新增了一行"add new line 4", 跟 (图 5) 中的 git diff 改动是一致的。

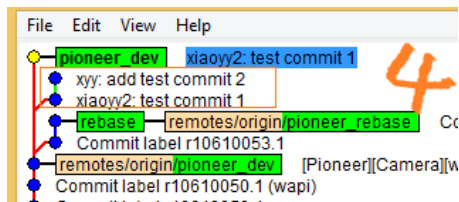




5.7 多次修改(git rebase -i)

如图在本地分支 `pioneer_dev` 上做了 2 次提交, 我想合并成一个提交可用 `git rebase -i HEAD~2` (如果要合并 3 个 commit 就用 `HEAD~3`), 然后会跳转到图 (1) 用 `vi` 进行编辑, 根据下面的提示可以看出 `pick` 是保留这个 commit, `reword` 是编辑 message 等等.. 在这里我们用 `fixup` 这个选项即可, 就是保留第二次 commit 的内容, 但是不要这个 commit 的 log, 也就是说和第一次的 commit 进行了合并, 把第二个 commit 行前的 `pick` 改成 `f` 如图 (2), 然后 `wq` 保存退出就回到了命令行窗口 (3), 提示 `successfully rebased and updated`, 这样我们在 `gitk` 中 `F5` 刷新就能看到之前的 2 个提交都没有分支跟踪了, `pioneer_dev` 分支也指向了合并后的新提交. 如图 (4), 如果你忘记在图 (2) 中修改 commit message, 也可以最后再执行一次 `git commit --amend` 修改.





5.8 Patch 相关操作

5.8.1 产生 patch (git format-patch, git diff >)

从某个 commit 往前生成 n 个 patch 的命令: `Git format-patch -n <SHA1>` , 会生成从 0001-开头.patch 结尾的 n 个 patch 文件

从当前 commit 生成 1 个 patch: `git format-patch -1`

```
[xiaoy2@XIAOY2-21 /c/workspace/P1/tools]$ git log -5 --one-line
5617b6a Person1: Add 2 testlines
e15b9c6 person2: Add one new line
13f9712 Fix for call GenPerillImagesSpkgs.bat
a689e67 fix build issue
949993c fix logic error
[xiaoy2@XIAOY2-21 /c/workspace/P1/tools]$ git format-patch -2 13f9712
0001-fix-build-issue.patch
0002-Fix-for-call-GenPerillImagesSpkgs.bat.patch
[xiaoy2@XIAOY2-21 /c/workspace/P1/tools]$ git format-patch -1
0001-Person1-Add-2-testlines.patch
```

如果想将还没 commit 的改动做成 patch 用 `git diff > test.patch`

发给别人拿到 patch 后可以用 `git apply test.patch` 得到你的改动

5.8.2 打 patch (git am/apply)

可以用 `git am *.patch` 一次打上所有 `git format-patch` 产生的一系列 patch

```
[xiaoy2@XIAOY2-21 /c/workspace/P1/tools]$ ls
0001-fix-build-issue.patch 0002-Fix-for-call-GenPerillImagesSpkgs.bat.patch build8926.bat
[xiaoy2@XIAOY2-21 /c/workspace/P1/tools]$ git am *.patch
Applying: fix build issue
Applying: Fix for call GenPerillImagesSpkgs.bat
[xiaoy2@XIAOY2-21 /c/workspace/P1/tools]
```

也可以分别打上某个 patch

```
[xiaoy2@XIAOY2-21 /c/workspace/P1/tools]$ ls
0001-fix-build-issue.patch 0002-Fix-for-call-GenPerillImagesSpkgs.bat.patch
[xiaoy2@XIAOY2-21 /c/workspace/P1/tools]$ git am 0001-fix-build-issue.patch
Applying: fix build issue
[xiaoy2@XIAOY2-21 /c/workspace/P1/tools]$ git am 0002-Fix-for-call-GenPerillImagesSpkgs.bat.patch
Applying: Fix for call GenPerillImagesSpkgs.bat
[xiaoy2@XIAOY2-21 /c/workspace/P1/tools]
```

5.9 Gitk 工具(gitk --all&)

5.9.1 Gitk 介绍:

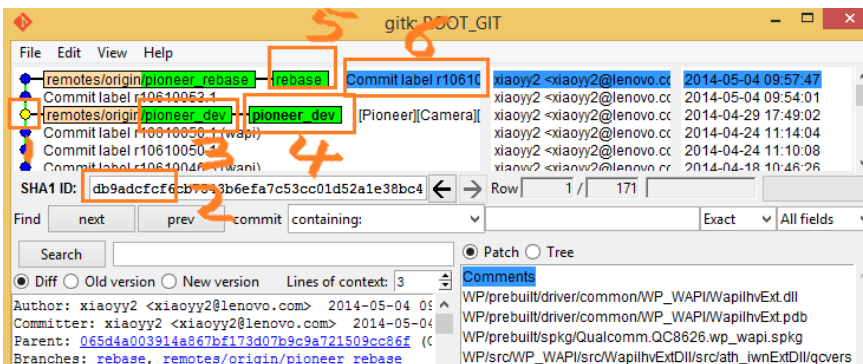
图中 (1) 处的小黄点代表 HEAD, 也就是当前 checkout 的点

(2) 处是鼠标选中点的 commit ID(也叫 SHA1), 如图 2 处的 SHA1 是 rebase 分支 (5) 所在点的 commit ID

(3) 和 (4) 有 2 个 pioneer_dev 分支, 前面带有 remotes/origin/ 的为远端分支(其中 origin 为远程仓库别名, 不同仓库名不同),

本图中有 2 个远程分支 pioneer_dev 和 pioneer_rebase, 2 个本地分支 pioneer_dev 和 rebase

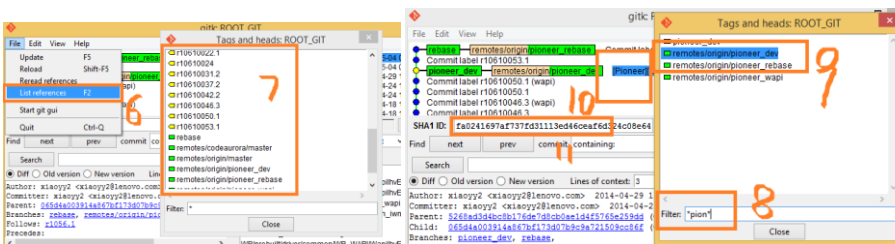
图中 (4) 处的分支名被高亮加粗显示, 代表当前 checkout 的分支是 pioneer_dev 分支

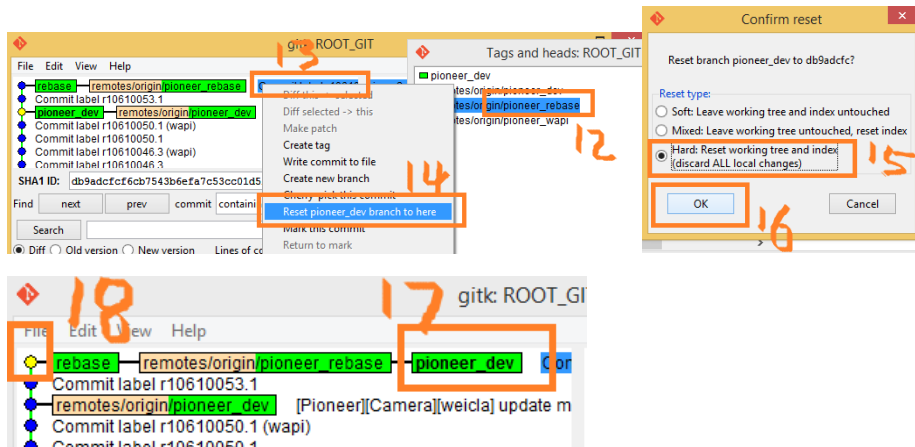


5.9.2 搜索分支/tag/commit, 并 checkout 到指定的点

在 gitk 界面按 F2 键 (6), 调出过滤窗口 (7), 在 filter 栏 (8) 输入 2 个星号, 中间输入分支/tag 关键字, 点击名字 (9), 该分支/tag 就会被 highlight 见 (10), 如果查找 commit, 直接在 (11) 处输入 SHA1 然后回车即可。

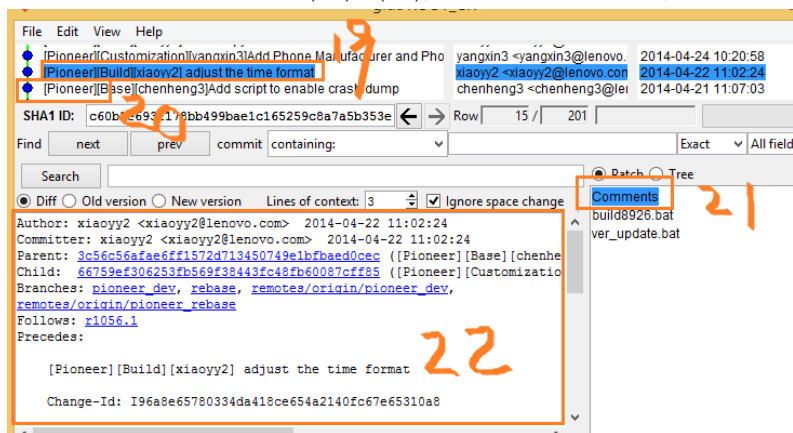
接下来我们选中 pioneer_rebase 分支 (12), 然后在高亮的 (13) 处右键, 选择 "reset ... to here" (14) 在弹出的窗口中选 "Hard..." (15), 然后 OK (16), 然后可以看到本地分支 pioneer_dev (17) 和小黄点 (18) 指向了 pioneer_rebase 所指向的 commit 点, 这样就完成了代码从 pioneer_dev 分支到 pioneer_rebase 分支的切换。

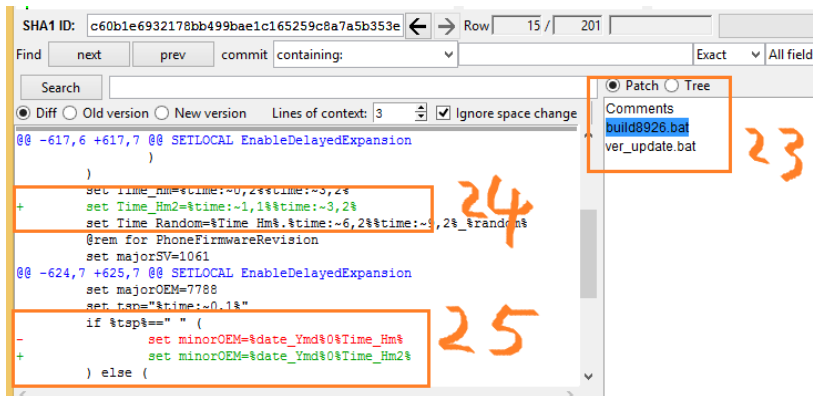




5.9.3 查看某个 commit 更新了什么

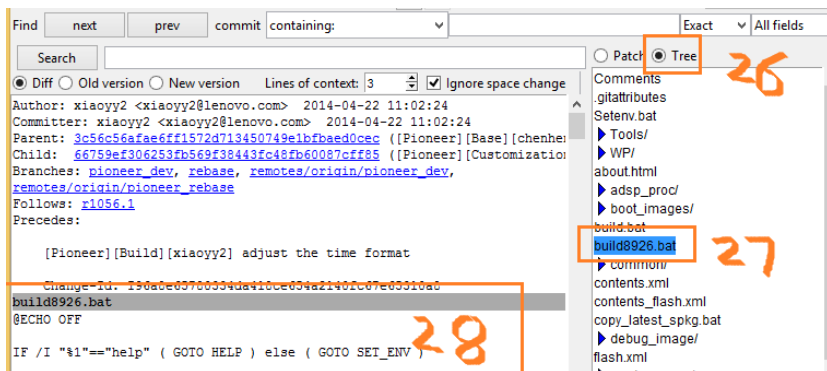
在 gitk 中任意点击一个 commit (SHA1 为 c60b1e) 的文字部分 (19)，高亮选中后显示的文件改动 (21) 和内容改动 (22) 都是当前这个 commit (19) 与上一个 commit (20) 之间的改动，本例中可以看到 c60b1e 这个 commit 共有 2 个文件不同 (23)，其中 build8926.bat 这个文件有 2 处改动 (24) 和 (25)，前有+号的绿色部分为新增行，前有-号的红色部分为删除行。





5.9.4 查看某个 commit 的完整内容

在选中“Tree” (26) 后，下面显示的就是 c60b1e 这个提交所包含的所有文件和目录，点击任意文件如 build8926.bat (27)，就会在左方 (28) 处显示这个文件的所有内容。



5.9.5 比较 2 个不相邻 commit 之间的差异

先选中一个 commit (29)，然后鼠标直接右键在另外一个 commit 的 message 处 (30)，选择 diff ...即可



5.9.6 查看某个目录/文件的改动

可以在 gitk 后跟上文件名, 或者目录名, 如:

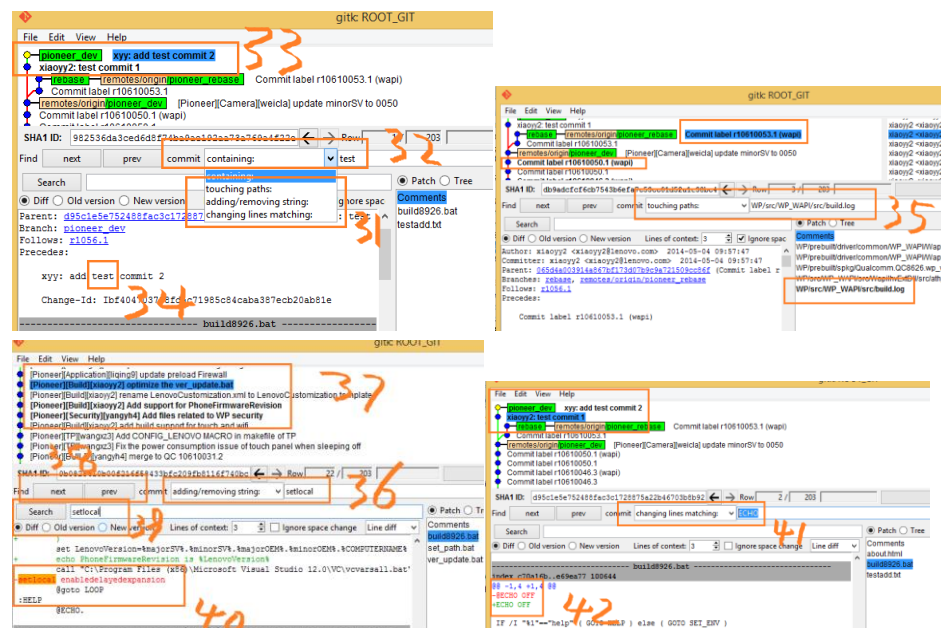
gitk build8926.bat 只会显示 build8926.bat 这个文件的改动相关的 commit.

gitk modem_proc 看到的只有这个文件夹的改动, 不在这个文件夹的改动都没有显示.

查看某两个 commit 直接某个文件/文件夹做了什么改动, 方法同查看 commit 间改动 (5.9.3).

5.9.7 搜索改动的文件/内容/注释

在 (31) 处选择 containing, 关键字填上 "test" 即可在所有的提交信息里搜包含有 "test" 字样的提交, 结果会加粗显示 (见 33 处)
若选 touching paths, 后面写上修改的路径和文件名 (35), 即可搜出所有修改过这个文件的 commit, 也会加粗显示
如果选择 adding/removing string, 后面跟上关键字 (36), 只会在所有 commit 的增减部分 (如图 40 处) 中搜这个关键字, commit 会高亮 (37), 可以在 (38) 点上一个下一个选择 commit, 在 (39) 处输入这个关键字在某个 commit 的内容中进行搜索具体位置. 选择 (41) changing lines matching 可以使用正则表达式搜索, 而且只要修改的行包含关键字即可搜到, 如 (42) 中的改动为删掉一个 @ 符号, 用 adding/removing string 必须搜 @ 这个字符才能搜到, 而用 changing lines matching 只要搜这行的任意字符都行 (包括使用正则匹配).



5.10 Git gui 工具

5.10.1 Git gui 之提交, 撤销, 修订

可以从 gitk 界面 File -> Start git gui (1), 也可以直接在命令行运行 git gui 启动.

下面我们新增一个文件，删除一个文件和修改一个文件，分别在命令行，gitk 和 git gui 中看看是什么状态。

在 (2) cmd 中输入 git status 查看修改情况，可见 about.html 标记为删除，build8926.bat 标记为修改，testadd.txt 标记为未跟踪，(此处代表新增文件)，如果要查看修改内容需要执行 git diff 查看

在 (3) gitk 中可以看到这个改动是用一个红点标记，能看到修改的文件和内容

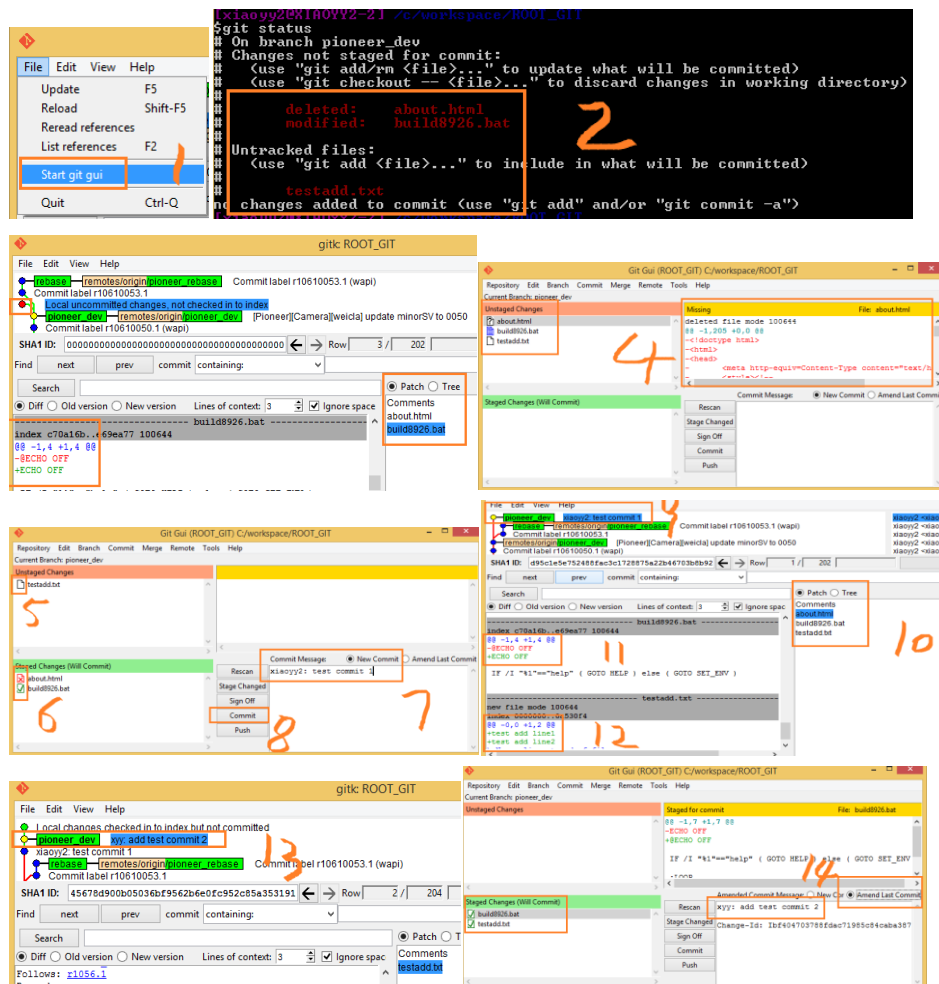
在 (4) git gui 中也能看到改动的文件和内容，下面提供了 stage 和 commit 等操作按钮。

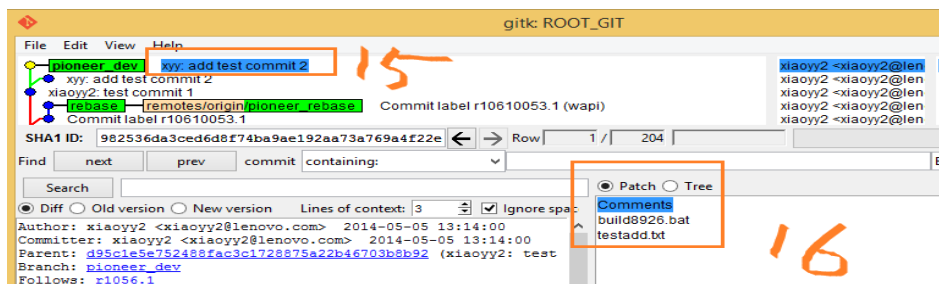
点击 (5) 文件前方的小图标，文件就被自动 stage 到 (6) 处，作用相当于 git add file 命令

点击 (6) 处文件前方的小图标，文件又被撤销回 (5) 处，作用相当于 git reset HEAD file...

在 (7) 处填写 commit message 后，点 (8) 进行 commit，就会生成一个提交 (9)

在 gitk 中我们可以看到这次修改的 commit 信息，文件列表和修改内容。接下来我们可以继续修改，继续提交生成第二个 commit (13)，然后我们发现第二个 commit 改的不对需要更新，可以先将修改的文件 add 到 (6) 处，然后选 (14) 对上次提交进行修订，这样产生了 commit (15)，其下方悬空的为之前的 commit (9)，被舍弃，修改文件列表只显示了最终的修改情况 (16)，这样就完成了对上次提交的修正。



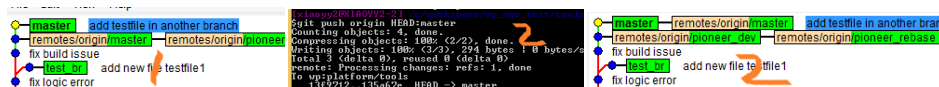


6 代码推送 (git push)

6.1 Push 的规则: 什么情况下可以推送,什么情况下不能

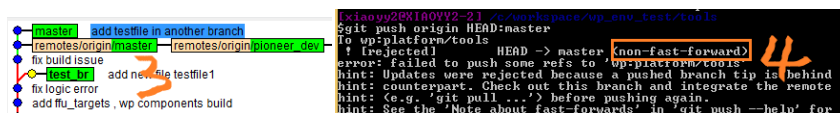
代码修改的前提是必须基于要提交分支在服务器端的最新点进行修改变, 然后才能推送到服务器, 否则会被服务器 reject.

如下图 (1) 中本地分支 master 基于远程分支 origin/master 分支做了一次提交, 然后可以执行 git push origin HEAD:master 就推送到远程的 master 分支了 (见图 2)。



如果不基于服务器端最新的点提交, 如下图 (3) 中将 test_br 分支的改动往 master 分支提交, 就会产生图 (4) 中 non-fast-forward 的错误。此处强调服务器端, 是因为有时虽然本地看到的是最新的点的改动, 但实际上服务器上又有其他人进行了提交, 这样也是提交不上去的, 解决的办法就是先执行 git pull --rebase <remote> <branch> 再执行 git push。

需要注意的是: 在推送给到 gerrit 上进行 code review 时不会提示有任何错误, 但是在 gerrit 网页上进行最终的 submit 操作时, git 服务器会对最新的提交点进行检测, 并反馈给 gerrit 失败信息 (见下文 “推送失败场景二”), 所以在执行 git push 操作前养成 pull + rebase 的好习惯, 并且在代码推送到 gerrit 上后督促 reviewer 尽快 submit, 避免中间有其他提交导致需要 rebase。



6.2 如何推送代码

6.2.1 直接推送 (refs/heads/<branch>)

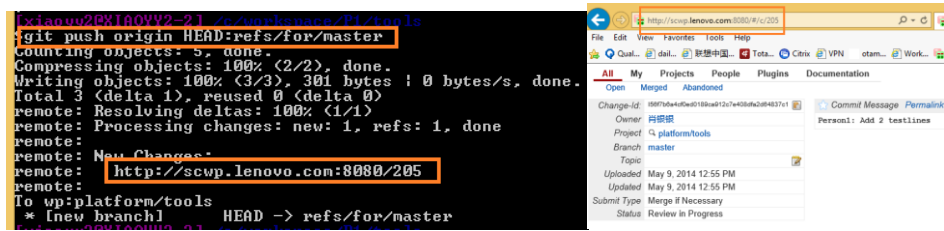
直接推送你的本地 commit 或 branch 到某个远端仓库 (需要有直接推送的权限)

命令: git push <repository> <refspec> 其中 <repository> 就是指 remote, 不指定默认为 origin; <refspec> 指明了本地的 src 和期望推到远端的 dest, 中间用冒号分隔, src 可以是分支名, 也可以是 SHA1, 我们一般用 HEAD, 也就是当前 checkout 的 commit

推到远端仓库, dest 一般是远端仓库的分支名, 或者 tag 名, 如 git push origin HEAD:refs/heads/pioneer_dev 就代表将你的 HEAD 所在的 commit 推送到 origin 这个仓库的 pioneer_dev 分支, 这里的 refs/heads/可以省略, 推送 br1 分支到 wp 这个远端仓库: git push wp br1, 推送 tag 到 origin 仓库: git push origin V1.1

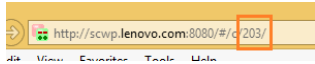
6.2.2 推送到 Gerrit 上进行 code review (refs/for/<branch>)

如果需要推送到 gerrit 上进行 code review, 需要将 refs/heads 替换成 refs/for
命令为 git push origin HEAD:refs/for/pioneer_dev
推送成功后会产生一个 gerrit 网址, 打开这个网址即可看到提交的内容,如:



6.2.3 更新 Gerrit 上正在 review 的 patch (refs/changes/<gerrit id>)

Gerrit ID 就是 gerrit link 中最后的数字编号, 如下图中的 203



patch 提交到 gerrit 进行 review 的过程中, 只要还没 submit, 就都可以持续的进行更新, 每次更新会产生一个新的 patch set, 如下图, 而产生 patch set 的方式, 就是把 push 到 gerrit 命令中的 refs/for/<branch>改成 refs/changes/<gerrit id>, 如 git push origin HEAD:refs/changes/203

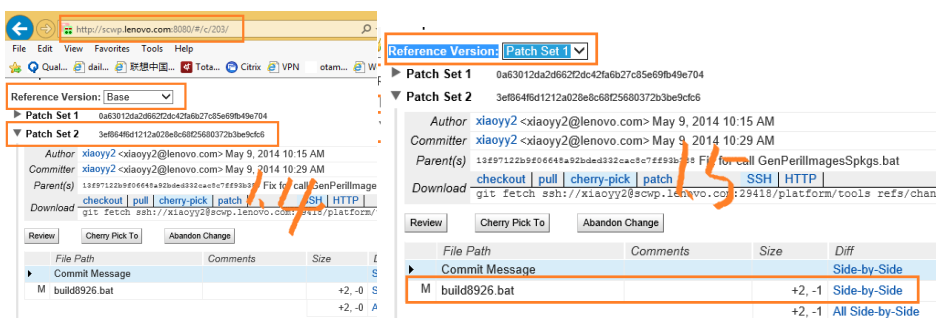
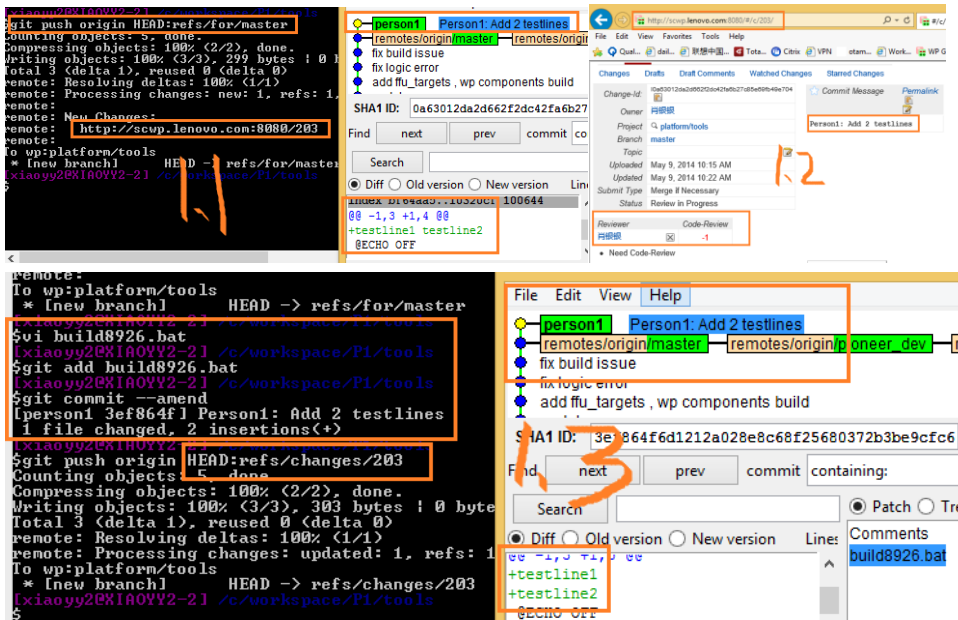


6.3 推送失败如何处理?

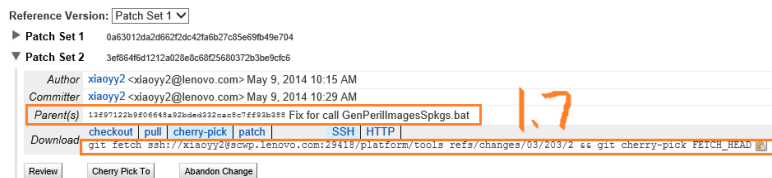
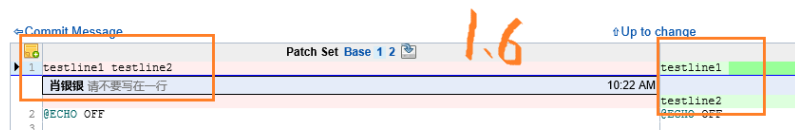
6.3.1 推送失败场景一：推送到 gerrit 上 review 的 patch 不合格,该如何更新?

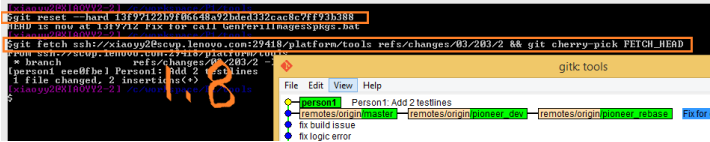
如图 (1.1) 所示 Person1 做了一个修改后提交到 gerrit 上进行 review, 在图 (1.2) 中进行 code review 时得到了一个不合格的负分, 需要修改后重新提交. 于是在图 (1.3) 中把这个修改更新后, 执行 git commit --amend, 然后再次提交到之前 review 的 gerrit 网址 <http://scwp.lenovo.com:8080/#/c/203/> 上去, 这时就需要用 git push origin HEAD:refs/changes/203 (如图 1.3) 对上个提交进行更新, 在 gerrit 上产生了 patch set 2 (如图 1.4), 默认的 Reference Version 是 base, 如 (图 1.5) 我们把它改

成 Patch Set1,再点击修改文件后的 Side-by-Side, 看到的的就是 patch set2 与 patch set 1 之间的更新如 (图 1.6), 这样就完成了对 review 不合格 patch 的更新, 如果想在本地更新 patch 的时候发现本地的提交已经被修改了,或者分支已经 hard reset 到其他地方了, 可以先 hard reset 回到 (图 1.7) 中 Parents 的 SHA1 处, 再执行下面 cherry-pick 的命令即可, 如 (图 1.8)



build8926.bat

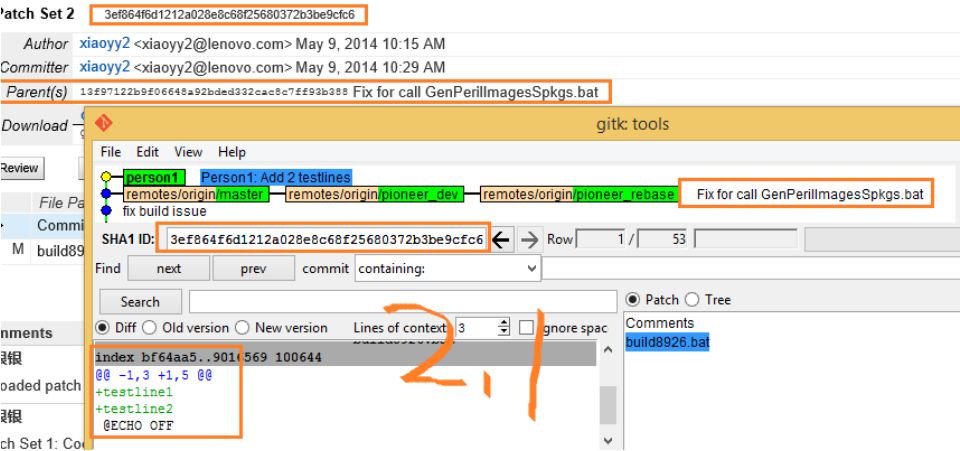


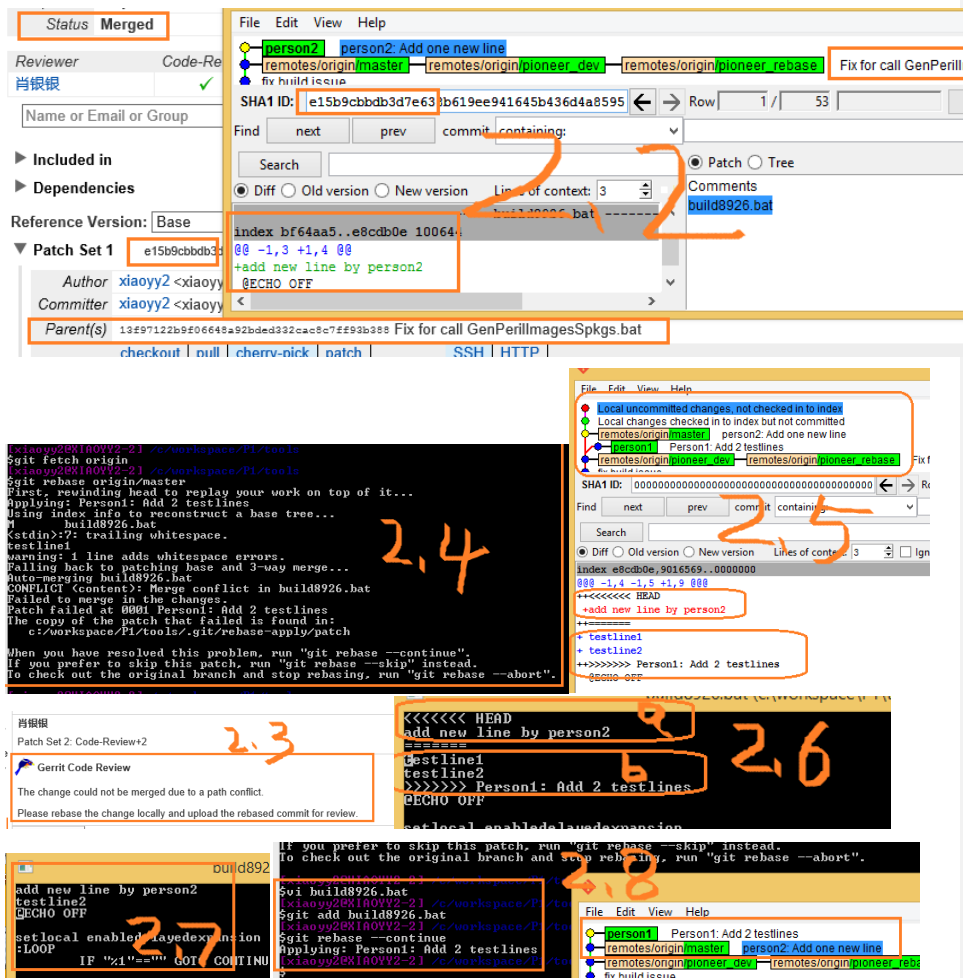


6.3.2 推送失败场景二：别人的提交先通过，我再提交 gerit 说有 merge 冲突

假设 person1 基于当前最新的 commit 提交了一个 patch1 到 gerit 上进行 review 如 (图 2.1), gerit ID 为 203
Patch1 在 review 期间 person2 也基于当前最新的 commit 提交了一个 patch2 到 gerit 上 (gerit ID 为 203), 并且很顺利的通过了 review 并 submit 了, 如 (图 2.2), 刚好这两个 patch 都修改到了同一个文件的同一行代码, 在 patch1 review 通过后进行提交的时候, 就会产生冲突提示 (图 2.3), 并要求 person1 解决冲突后重新提交如.
在此情况下, person1 需要先执行 git pull --rebase origin master (等同于 git fetch origin && git rebase origin/master) (图 2.4) 这个命令会先下载 patch2 到本地, 然后将 patch2 rebase 到已经提交的 Patch1 这个 commit 上, 列出冲突 (图 2.5) 等 person1 解决后再进行提交. 解决冲突步骤如下:

1. 编辑冲突文件 (如图 2.6), 对冲突部分进行取舍 ("<<<<<<" 号与 ">>>>>>" 号所在行之间的内容为冲突的部分, 其中 a 区域为 person2 提交的 patch2 的内容, b 区域为 person1 提交的 Patch1 的内容, 中间用 "=====" 分隔,) 并将 git 标记冲突的符号 "<<<<<<", "=====", ">>>>>>" 所在的行删除, 假如最终的内容是图 (2.7) 的样子.
2. 我们:wq 保存退出后, 在命令执行图 (2.8) git add 和 git rebase --continue
3. 最后用 git push origin HEAD:refs/changes/203 对之前的 patch 进行更新, 这个冲突就解决完成了.





6.3.3 推送失败场景三：当前 patch 所依赖的 patch 尚未提交

如图 (3.1) 所示本地进行多次 commit 后执行 git push, 就会将所有的本地分支都推送到 Gerrit 上, 每个 commit 产生一个 Gerrit link, 这些 Gerrit 之间会用图 (3.2-3.4) 中的 "Depends On" 和 "Needed By" 标识互相的依赖关系, 在 gitk 中可以看出先提交的 "First commit" 会 **Needed By** 后提交的 "Second commit", 后提交的 "Second commit" 会 **Depends On** "First commit".

在提交这 3 个 patch 的时候需要按 commit 的顺序依次 submit, 否则对其他未提交的 commit 有依赖的 patch 会出现 "Submitted, Merge Pending" 的状态, 直至它依赖的 patch 变成 submitted 状态. 如我先 review 了第二个 (图 3.3) 和第三个提交 (图 3.4), 但是由于他们依赖的第一个提交还未 merged (图 3.2), 所以这俩提交都只是变成了 "Submitted, Merge Pending" 的状态, 没有 submit 到 git 远端仓库中. 在 (图 3.5) 中我对第一个提交进行了 submit 操作, 第二个和第三个提交的状态立刻也变成了 Submitted 状态 (图 3.6), 3 个提交都 submit 到 git 远端仓库中去了.


```
git push origin HEAD:refs/for/master
Counting objects: 11, done.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (9/9), 763 bytes | 0 bytes/s, done.
Total 9 (delta 3), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3)
remote: Processing changes: new: 3, refs: 3, done.
remote: New Changes:
remote: http://scup.lenovo.com:8080/200
remote: http://scup.lenovo.com:8080/201
remote: http://scup.lenovo.com:8080/202
To up:platform/tools
 * [new branch] HEAD -> refs/for/master
[xiaoyy2@X180VY2-21 ~]$ git tools
File Edit View Help
 * person1 xiaoyy2: Third commit
 * xiaoyy2: Second commit
 * xiaoyy2: First Commit
 * remotes/origin/master remotes/origin/master
 * fix build issue
SHA1 ID: 5aa343c3757753a0a2c4ce95dc95d70f1ead1d3
```

Owner 肖根根

Project platform/tools

Branch master

Topic

Uploaded May 9, 2014 9:40 AM

Updated May 9, 2014 9:40 AM

Submit Type Merge if Necessary

Status Review in Progress

xiaoyy2: First Commit

Need Code-Review

Name or Email or Group Add Reviewer

Dependencies

Subject Owner Project Branch Updated

Depends On (None)

Needed By

xiaoyy2: Second commit (SUBMITTED) 肖根根 platform/tools master 9:45 AM

Reference Version: Base

Owner 肖根根

Project platform/tools

Branch master

Topic

Uploaded May 9, 2014 9:40 AM

Updated May 9, 2014 9:45 AM

Submit Type Merge if Necessary

Status Submitted, Merge Pending

xiaoyy2: Second commit

Reviewer Code-Review

肖根根

Name or Email or Group Add Reviewer

Dependencies

Subject Owner Project Branch Updated

Depends On

xiaoyy2: First Commit 肖根根 platform/tools master 9:40 AM

Needed By

xiaoyy2: Third commit (SUBMITTED) 肖根根 platform/tools master 9:45 AM

Owner 肖根根

Project platform/tools

Branch master

Topic

Uploaded May 9, 2014 9:40 AM

Updated May 9, 2014 9:45 AM

Submit Type Merge if Necessary

Status Submitted, Merge Pending

xiaoyy2: Third commit

Reviewer Code-Review

肖根根

Name or Email or Group Add Reviewer

Dependencies

Subject Owner Project Branch Updated

Depends On

xiaoyy2: Second commit (SUBMITTED) 肖根根 platform/tools master 9:45 AM

Needed By (None)

Owner 肖根根

Project platform/tools

Branch master

Topic

Uploaded May 9, 2014 9:40 AM

Updated May 9, 2014 10:00 AM

Status Merged

xiaoyy2: First Commit

Reviewer Code-Review

肖根根

Name or Email or Group Add Reviewer

Included in

Dependencies

Subject Owner Project Branch Updated

Depends On (None)

Needed By

xiaoyy2: Second commit (MERGED) 肖根根 platform/tools master 10:00 AM

Owner 肖根根

Project platform/tools

Branch master

Topic

Uploaded May 9, 2014 9:40 AM

Updated May 9, 2014 10:00 AM

Status Merged

xiaoyy2: Second commit

Reviewer Code-Review

肖根根

Name or Email or Group Add Reviewer

Included in

Dependencies

Subject Owner Project Branch Updated

Depends On

xiaoyy2: First Commit (MERGED) 肖根根 platform/tools master 10:00 AM

Needed By

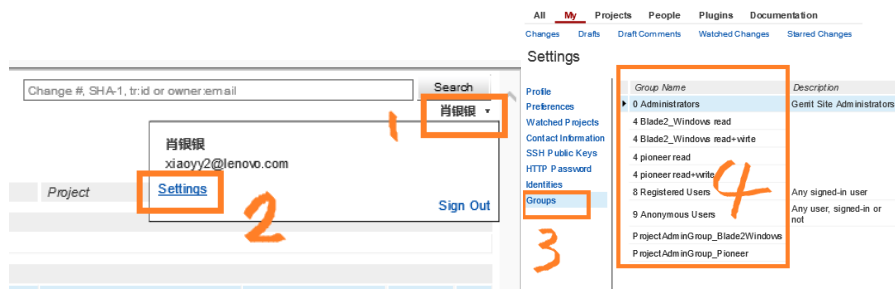
xiaoyy2: Third commit (MERGED) 肖根根 platform/tools master 10:00 AM

6.4 如何使用 Gerrit 网页查看推送结果

6.4.1 查看自己的设置/权限

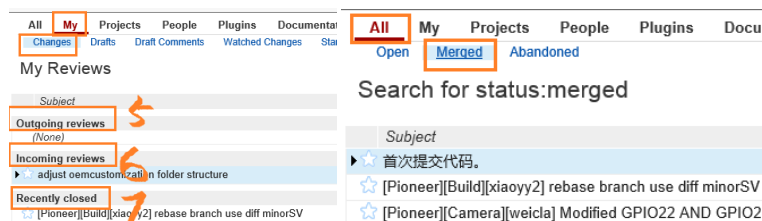
打开 Gerrit 网页，登录后在右上角点击自己的名字 (1)，选设置 (2)，然后在左边选组后就可以看到自己所属的权限组。

一般 read 只有 review+/-1 权限，可以下载代码，也可以推送代码到 Gerrit 上，read+write 除了 read 权限外，还可以在 Gerrit 上进行 review +/-2 和 submit 权限。



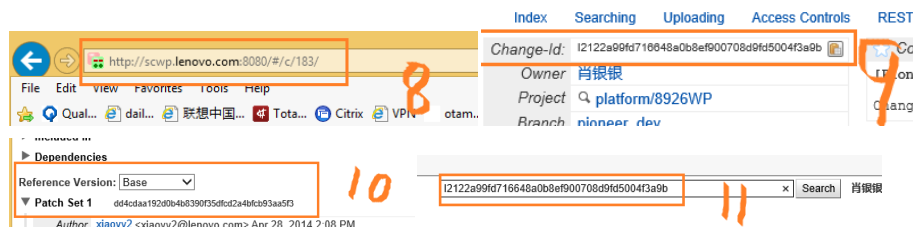
6.4.2 查看 patch (open/merged/abandon/my changes)

打开 Gerrit 网页并登录后，可以通过上方的 My → Changes 查看跟自己相关的提交，其中 outgoing reviews (5) 是自己提交 Open 状态的，Incoming reviews (6) 是别人提交加了自己 review 的，Recently closed (7) 是自己提交但是状态已经关闭的。同时还可以查看所有人的提交，从 All → Open/Merged/Abandoned 进行查看。



6.4.3 搜索 patch (status,branch,project,owner,time)

可以在搜索栏 (11) 中输入 Gerrit ID (8)，或者 Change ID (9)，或者 commit id (10) 直接跳转到相关 Gerrit 页面。也可以用 status, branch, owner, age 等去过滤要搜索的 patch 如 (12)，更多用法见 Gerrit 网页中 Document → Searching。

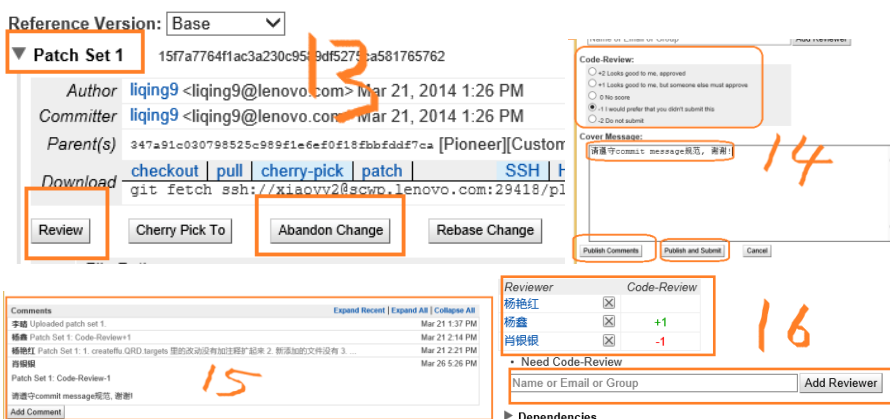




6.4.4 Review patch (add reviewer,+/-1, +/2, Abandon,submit)

Review 后在相应的 patch set 下 (13) 选择 Review 进行打分和提交 (14), 只有同时满足 Code-Review +2, 和点击了 Publish and Submit, 在没有 merge 冲突的情况下才能将这个 patch 提交到 git 远程仓库里. 如果 patch 没有通过 review 可以用-1 或者-2, 并说明自己的意见后, 点击 Publish comments 即可, 如果 patch 不允许提交, 可以直接在 (13) 中 Abandon Change 舍弃. Review 的记录和 comments 会在 patch 下方显示 (15), 打分结果在 (16) 处显示, 如果需要其他人 review 可以在 Add Reviewer 处输入邮箱即可.

► Dependencies



6.4.5 从 Gerrit 上下载 patch

当一个 patch 还在 review 状态时, 别人是无法从 git 仓库里看到的, 这样就需要我们从 gerrit 上获取

首先需要查看这个提交的 git 库名, 进入这个 git 库后找到 Parents 这个提交, 基于这个点(也可以不基于 parents 点)进行 cherry-pick, 先 checkout 一个本地分支, 然后将下面的命令粘贴到命令行窗口, 执行成功后本地就获取到了这个 patch.

Projectplatform/8926WP

Branchpioneer_dev

Topic

UploadedMar 21, 2014 1:37 PM

UpdatedMar 26, 2014 5:26 PM

Submit TypeRebase if Necessary

StatusReview in Progress

Reviewer	Code-Review
苏艳红	<input checked="" type="checkbox"/>
苏鑫	<input checked="" type="checkbox"/> +1
苏银银	<input checked="" type="checkbox"/> -1

Need Code-Review

Dependencies

Reference Version:Base

Patch Set 115f7a7764f1ac3a230c9589df5275ca581765762

Author	liqing9 <liqing9@lenovo.com>	Mar 21, 2014 1:26 PM
Committer	liqing9 <liqing9@lenovo.com>	Mar 21, 2014 1:26 PM
Parent(s)	247a91c030798525c989f1e6ef0f19fbbfdd7ca Pioneer[Customization][yangyh4] Remove some commented lines which cause ...	
Download	checkout pull cherry-pick patch SSH HTTP	
	git fetch ssh://xiaoyy2@scwp.lenovo.com:29418/platform/8926WP refs/changes/06/106/1 && git cherry-pick FETCH_HEAD	

7 Git 小技巧

7.1 Git 命令自动补全 (git-completion.bash)

将这个文件放到 git 的 home 目录, 然后在 .bashrc 中加上 source ~/git-completion.bash

在输入 git 命令时按 tab 键即可自动补全支持的命令.

<https://github.com/git/git/blob/master/contrib/completion/git-completion.bash>

7.2 修改 git 命令行提示符颜色和格式 (git ps1)

在 .bashrc 中添加一行 export PS1="\033[0;35m[\u@\h]\033[m \033[0;34mlw\033[m\n\$"

效果为: 

参考 <http://code-worrier.com/blog/git-branch-in-bash-prompt/>

7.3 Git 别名 (git alias)

可以在 .gitconfig 文件包含了这样一些常用命令的别名:

[alias]

st = status

co = checkout

cm = commit

psuh = push

有些程序员可能会有某几个单词敲的特别不习惯，比如总把 push 敲成 psuh，那索性就可以把 psuh 作为 push 的一个别名加到配置文件中，多省事。

如果你碰巧忘记了自己定义的别名，可以通过下面这个命令来快速查阅你的配置文件：

git config -l

7.4 更多小技巧

参考: <http://www.linuxeden.com/html/develop/20100612/103365.html>