## 程建给的git命令建议:

git 常用命令

* git clone XXX//克隆一份代码到本地仓库
* git pull //把远程库的代码更新到工作台
* git pull --rebase origin master //强制把远程库的代码跟新到当前分支上面
* git add . //把本地的修改加到stage中
* git commit -m 'comments here' //把stage中的修改提交到本地库
* git push //把本地库的修改提交到远程库中
* git branch -r/-a //查看远程分支/全部分支
* git checkout master/branch //切换到某个分支
* git checkout -b test //新建test分支
* git checkout -d test //删除test分支
* git merge master //假设当前在test分支上面，把master分支上的修改同步到test分支上
* git stash //把未完成的修改缓存到栈容器中
* git stash list //查看所有的缓存
* git stash pop //恢复本地分支到缓存状态
* git blame someFile //查看某个文件的每一行的修改记录（）谁在什么时候修改的）
* git status //查看当前分支有哪些修改
* git log //查看当前分支上面的日志信息
* git diff //查看当前没有add的内容
* git diff --cache //查看已经add但是没有commit的内容
* git diff HEAD //上面两个内容的合并
* git reset --hard HEAD //撤销本地修改
* ls –al 查看当前目录下面所有的文件夹
* vim 文件名 编辑文件

git 回退本地版本:

git reset --hard 9d85ad9a (后面一段是git log查看历史记录中的版本唯一标示)回退本地版本,回退完之后可以使用git log查看本地的代码是否回退成功.

Git reset –hard HEAD^ 往前回退一个版本.

Git reset –hard HEAD^^ 往前回退两个版本.

Git 回退远程版本;(以下两个命令随便使用一个即可.前提是本地的版本已经回退.)

git push origin HEAD --force #远程提交回退

git push -f强制回退远程仓库到当前版本 (一般会有如下提示:因为该分支是受保护分支,所以不被允许.)

$ git push -f

Total 0 (delta 0), reused 0 (delta 0)

remote: GitLab: You are not allowed to force push code to a protected branch on this project.

To https://10.0.69.22:2201/crm/jscrm.git

! [remote rejected] dev -> dev (pre-receive hook declined)

error: failed to push some refs to 'https://os-houtt@10.0.69.22:2201/crm/jscrm.git'

Git merge –abort

Git Merge过程中,如果想废弃本册merge操作,可以使用如上命令.

将远程git仓库里的指定分支拉取到本地（本地不存在的分支）

git checkout -b 本地分支名 origin/远程分支名

推送本地分支到远程仓库

git push --set-upstream origin 分支名

删除远程分支

Git push origin –delete 分支名

远程分支重命名

1. 先把本地分支更新为最新代码;然后删除远程分支: git push origin –delete 远程分支名
2. 重命名本地分支:git branch –m 旧分支名 新分支名
3. 把本地分支推送到远程: git push –set-upstream origin 远程分支名字;

Git log 查看提交文件记录

查看提交记录: Git log

查看前三次提交的文件记录: git log -3 –stat

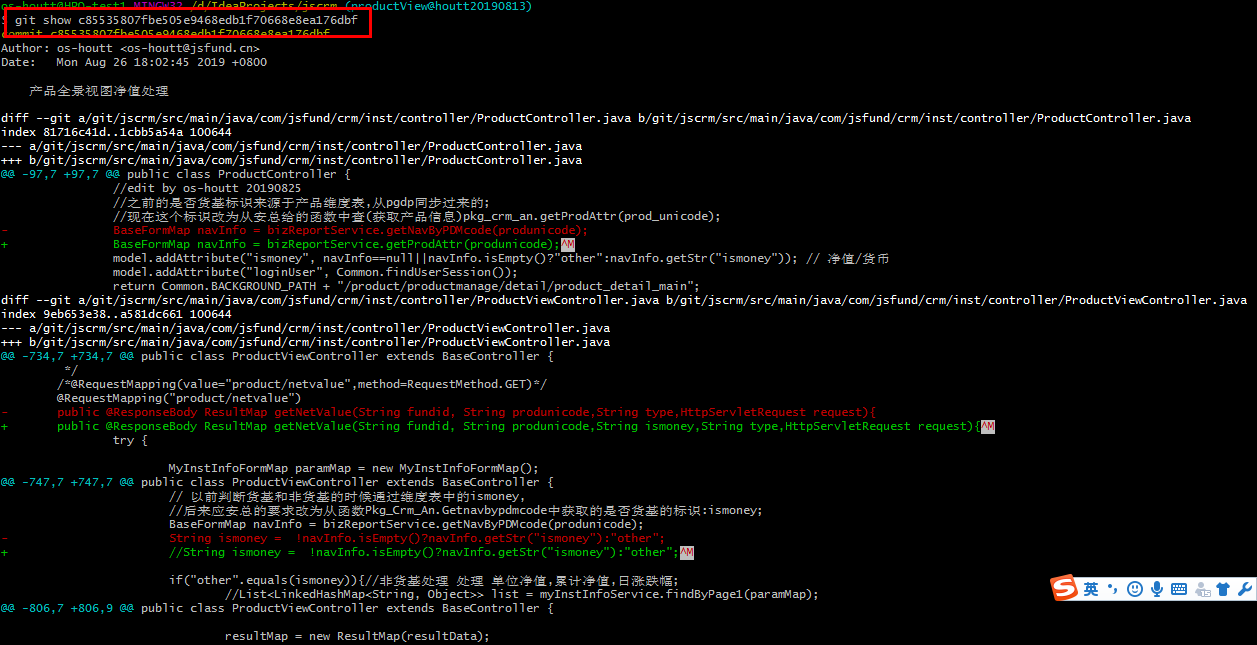
查看某一此提交的文件记录:

例子:



如果要 查看以上提交记录中第二次提交的内容:

使用 git show commit的唯一标示 –stat (注:--stat加不加都可以)



Git 强制忽略本地修改的文件:

git update-index --assume-unchanged git/jscrm/.gitignore(路径加文件名)

git branch --set-upstream 本地关联远程分支

最近使用git pull的时候多次碰见下面的情况：

There is no tracking information for the current branch.

Please specify which branch you want to merge with.

See git-pull(1) for details.

git pull <remote> <branch>

If you wish to set tracking information for this branch you can do so with:

git branch --set-upstream-to=origin/<branch> release

其实，输出的提示信息说的还是比较明白的。

使用git在本地新建一个分支后，需要做远程分支关联。如果没有关联，git会在下面的操作中提示你显示的添加关联。

关联目的是在执行git pull, git push操作时就不需要指定对应的远程分支，你只要没有显示指定，git pull的时候，就会提示你。

解决方法就是按照提示添加一下呗：

git branch --set-upstream-to=origin/remote\_branch your\_branch

其中，origin/remote\_branch是你本地分支对应的远程分支；your\_branch是你当前的本地分支。

我整理了一下流程和命令，建议大家使用命令行 :）

举例：

我接到了一个添加页面的任务，操作如下

git checkout master\_new  //切换到生产环境分支

git pull  //拿到生产环境最新代码

git checkout -b addPage@cj20181026  //切出一个自己的分支开始工作

代码开发完成后

git add  addPage.jsp  // add 自己修改的代码

git commit -m"添加页面" // commit 到本地的git仓库

测试环境测试

git checkout acceptance //切换到测试环境分支

git  pull   //拿到测试环境最新代码

git merge addPage@cj20181026 //把自己的分支合并到测试环境

如遇到冲突，会有提示, 如下

Auto-merging addPage.jsp

CONFLICT (content): Merge conflict in addPage.jsp

Automatic merge failed; fix conflicts and then commit the result.

这个时候如果你执行 git status时，会有提示，如下

You have unmerged paths.

(fix conflicts and run "git commit")

(use "git merge --abort" to abort the merge)

Unmerged paths:

(use "git add <file>..." to mark resolution)

both modified: addPage.jsp

no changes added to commit (use "git add" and/or "git commit -a")

Git用<<<<<<<，=======，>>>>>>>标记出不同分支的内容，所以找到冲突文件后，直接搜索 <<<<就能快速定位冲突点，但要注意，不一定只有一个地方冲突，别漏掉（如不知如何解决冲突，或解决时遇到问题，欢迎来扰）

解决后

git add  addPage.jsp  // add 自己修改的代码

git commit -m"for merge" // commit 到本地的git仓库

git push

如果在测试时发现bug，请回到自己的分支修改，再merge回acceptance，保证自己分支是完好的，代码是全的

生产环境

git checkout master\_new //切换到测试环境分支

git  pull   //拿到测试环境最新代码

git merge addPage@cj20181026 //把自己的分支合并到测试环境

//如遇冲突查看测试环境

git push

一旦merge到master\_new后，你的分支的生命周期就结束了，如果生产环境要修改，请再次从master\_new切分支来解决

如果需要在同一分支协作开发时

git push -u origin addPage@cj20181026 //把自己的分支 push到服务器

协作同事操作

git branch -r  查看服务器分支列表

git checkout addPage@cj20181026 //拿到协作分支

-----------------------------------------------------------------------------

Git fetch

1. git fetch origin branch1:branch2

上面的fetch操作，使用远程branch1分支在本地创建branch2(但不会切换到该分支),如果本地不存在branch2分支, 则会自动创建一个新的branch2分支,

1. git fetch origin branch1

从远程的origin仓库的branch1分支下载代码到本地的origin branch1

## Git pull和git fetch的区别:

1. git fetch 相当于是从远程获取最新到本地，不会自动merge，如下指令：

git fetch orgin master //将远程仓库的master分支下载到本地当前branch中

git log -p master ..origin/master //比较本地的master分支和origin/master分支的差别

git merge origin/master //进行合并

也可以用以下指令：

git fetch origin master:tmp //从远程仓库master分支获取最新，在本地建立tmp分支

git diff tmp //將當前分支和tmp進行對比

git merge tmp //合并tmp分支到当前分支

2. git pull：相当于是从远程获取最新版本并merge到本地

git pull origin master

git pull 相当于从远程获取最新版本并merge到本地

在实际使用中，git fetch更安全一些

## Git rebase(\*\*\*\*\*)

正常的解决冲突过程是:

1，git add .

2，git commit -m "..."

3，git push时因为本地仓库代码与远程仓代码有冲突，所以接下来

4，git pull拉取远程代码，而冲突需要手动解决

5，解决好后重新进行git add . git commit -m".." git push

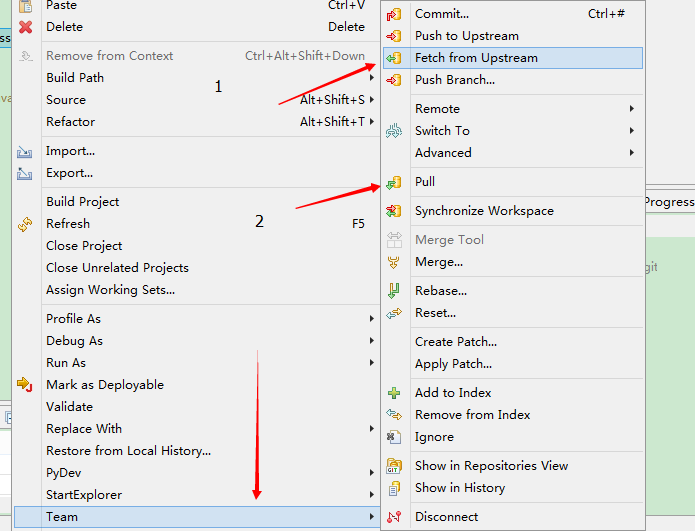
而git pull 这一步如果加上了 --rebase的选项，那么第5步操作将变成如下

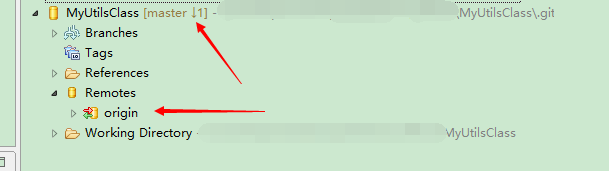
git add .

git rebase --continue

git push

Git提交代码的步骤:

1. 先更新 "远程服务器 --> 本地服务器"，再进行 更新 " 本地服务器 --> 本地代码"。
2. Eclipse上的操作步骤:
   1. 
   2. 1.Fetch from Upstream:代表的意思是只更新 "远程服务器 --> 本地服务器" 简称(Fetch);



当你更新后，你当前更新的项目与Git Repositories都会有类似一个向下的箭头，这代表“远程服务器”有东西更新到你的“本地服务器 ”，具体数量就是箭头隔壁的数字

2.指向代表的意思是 "远程服务器 --> 本地服务器" 与 " 本地服务器 --> 本地代码" 都执行 简称(Pull)

(当然有先后顺序的，先 "远程服务器 --> 本地服务器" 后 " 本地服务器 --> 本地代码")

这个操作其实是面向懒人把Git当成SVN使用，一步更新，官方也是不推荐这样做，因为这样，你本地代码，可能一下子代码冲突很多，给开发人员带来很多不便。

所以还是老实点，一步步来更新吧。

3.说完更新，接着说提交代码。

其实经过刚才的更新讲解后，小伙伴们都知道"远程服务器 --> 本地服务器" 与 " 本地服务器 --> 本地代码" 的关系了，

所以提交代码实质上都要走这样的操作 " 本地代码 --> 本地服务器" "本地服务器 --> 远程服务器"

还是上图吧！

其实小伙伴就记住两个词就好了。

Commit : " 本地代码 --> 本地服务器"

Push : "本地服务器 --> 远程服务器"

通常 Push 不了，无非就两个问题。

第一。你代码冲突还没有解决，一会 “代码冲突” 说具体讲解

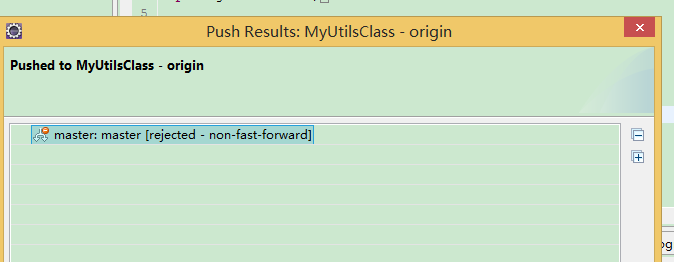
第二。就是可恶的网络问题。(同学。这个我可帮不了你了)

## Git冲突的问题

如果你是先修改完代码还没有commit，在执行Fetch,那你还没有feel到代码冲突。

I.只要你commit到本地服务器，IDE编译器就自动告诉你，已经冲突，要求你立刻解决。

II.如果你比较任性的话。直接想Commit And Push, IDE编译器也会很任性的回复你这样的图



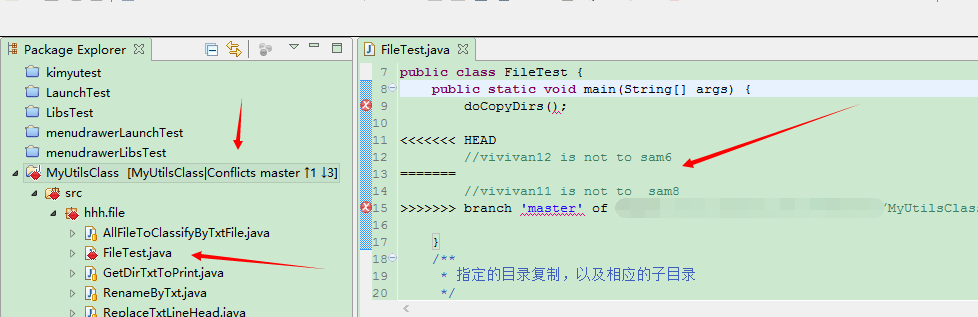
拒绝你Push，说你还有问题没有解决。

但是你已经Commit了，但是又无法Push，

这是你应该要执行一下 Pull ，把 刚才Fetch 的代码植入"本地代码",这时IDE编译器才自动告诉你，已经冲突，要求你立刻解决。

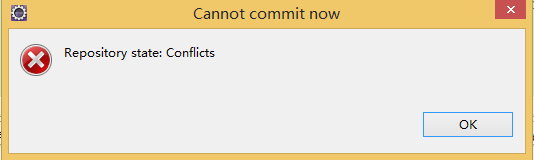
并且把冲突文件标记为一个双向箭头

如图

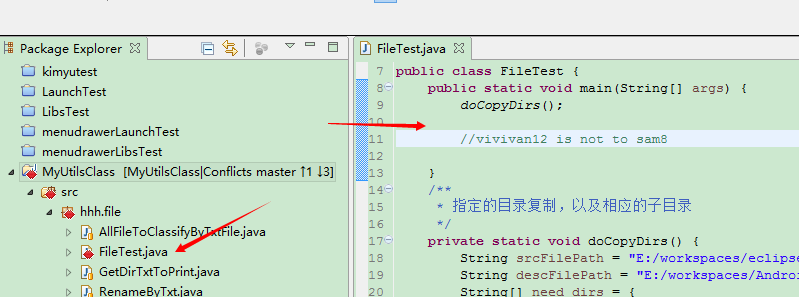


如果你现在还想执意Push代码，得到的就是下图:

Repository state ： Conflicts（简单意思就说：大哥，代码冲突啊，解决后，再提交吧）

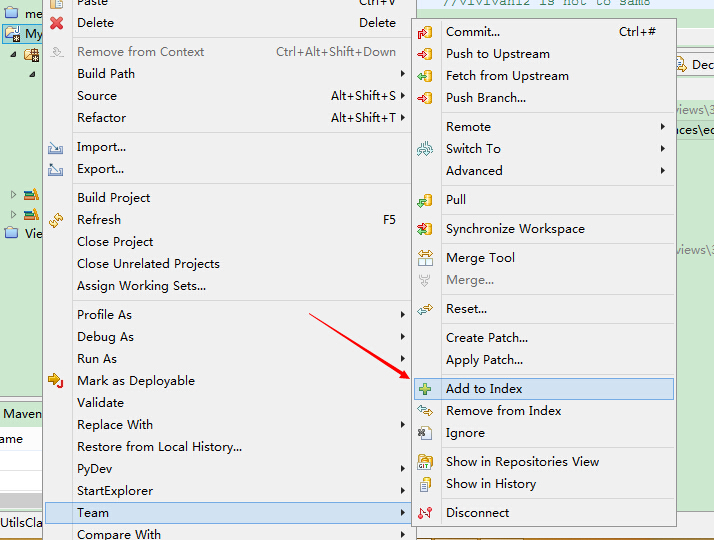


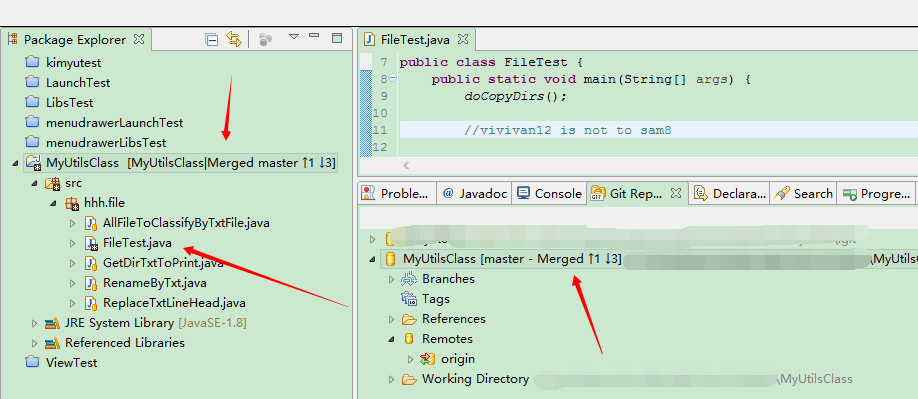
然后，你解决后，保存，发现，那个文件还是像下图的那样，无法提交。



这时，应该

执行这样“Add to Index ”的操作，告诉IDE编译器你已经解决冲突了，如下图：





现在你就可以去Commit 也好，Commit And Push 也好，IDE编译器都会通过的（其实告诉你。去吧，骚年！）

通过点击Merge能清楚知道，你的代码，解决后冲突的代码，服务器的代码

解决后冲突的代码直接覆盖你的本地代码。(或者你点击Close，然后冲突的代码直接跳在你的本地代码，让你在本地代码修改。)

解决完冲突后，就可以Commit和Push了。比Eclipse方便一点点。

Git bash here客户端使用记录:

1. 首先右键git bash here 的时候,要通过cd转到项目所在的目录下面,要不使用git命令无效.
2. 当使用git branch –a的时候(该命令是用来查看远程和本地的所有分支),使用Q键来跳出.
   1. git branch查看本地所有分支
   2. git branch -r查看远程所有分支
   3. git branch -a查看本地和远程所有分支

新建分支 git branch [-f] <branchname>：

新建一个分支，但不切换。如图新建了test分支，但是目前依旧在dev5.7分支上。

你可以使用 git checkout -b <branchname> 新建并切换至新分支

三、删除分支 git branch (-d | -D) <branchname>：

1.git branch -d <branchname>删除本地分支，其中<branchname>为本地分支名

2.git branch -d -r <branchname>删除远程分支，其中<branchname>为本地分支名

删除后，还要推送到服务器上才行，即git push origin :<branchname>

如图删除本地test分支，使用-D则表示强制删除，相当于 --delete --force

四、重命名分支 git branch (-m | -M) <oldbranch> <newbranch>：

如图重命名oldbranch为newbranch，使用-M则表示强制重命名。

如果你需要重命名远程分支，推荐的做法是：

删除远程待修改分支

push本地新分支名到远程

五、git 中一些选项解释

-d  --delete：删除

-D  --delete --force的快捷键

-f  --force：强制

-m  --move：移动或重命名

-M  --move --force的快捷键

-r  --remote：远程

-a  --all：所有

# Git 中 git checkout -- <file>的真正用法

网上关于这个撤销指令的教程多的数不胜数，但是在自己学习中还是遇到一些坑，在此记录下来，也给同样遇到坑的同学一点参考

先说遇到问的问题吧 在廖神的博客里 对于这个指令的解释是



Q2.png

但后来我在CSDN的一篇文章中见到的解释是



Q1.png

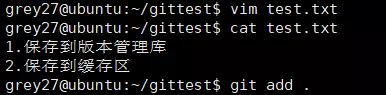
这就出现了两种意思 在我修改了工作区的内容后 使用 git checkout -- <file>到底是从缓存区还是版本库中还原呢 ，对此只有实践才有发言权

第一步：在管理库中添加一个版本：



1.png

第二步：在缓存区中添加一个版本：



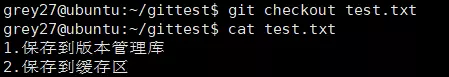
2.png

第三步：修改工作区文档



3.png

第四步：第一次使用git checkout -- <file>：



4.png

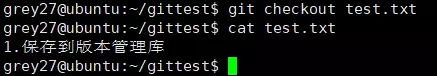
好了这时还原的结果是从缓存区里拉取的还原，貌似CSDN的网友说的更对。别急现在还不能下结论，毕竟廖神怎么会犯这种低级错误呢

第五步：清空缓存区：



5.png

第六步：这个时候缓存区没有版本了再次使用git checkout -- <file>：

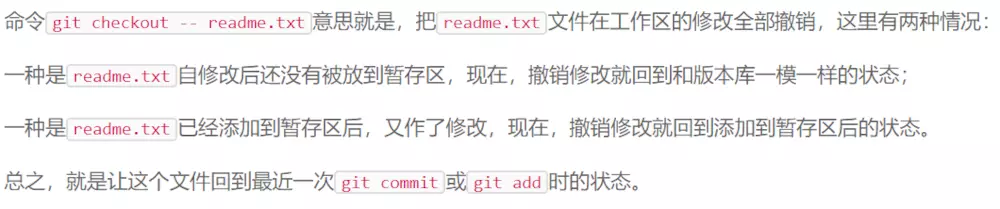


6.png

惊喜的事情发生了，工作区的版本竟然从版本库中还原了！

结论

通过这六步操作 我觉得git checkout -- <file>指令应该是从先从缓存区中拉取版本还原，如果没有再到版本库中拉取还原。在之后重新翻看廖神博客时，发现廖神在这句话的前一章对这句指令的解释是



7.png

（跟我最后得出的结论一样了，毕竟commit后自然没有缓存区版本了）

最后细心的同学应该一开始就发现了我的错误，对的 我使用的指令是 git checkout <file>原因是刚开始我使用git checkout --<file>语句一直报错，注意了--后面我没有加空格 因为之前学习的指令--后都是直接加指令的 ，最后无奈的取消了--，但一套指令下来我觉得和加了--的效果一样 而且后来也找到了廖神对这个指令的正确解释也就不加以修改了，我相信这个错误不止我一个小白出现过吧，在百度git checkout --<file>指令错误时没有搜到指正这个问题的贴子（可能是我搜索功力还不够吧）所以尽管我闹出了这个笑话 我还是决定把它写出来，给之后遇到这个问题的同学一点参考吧

## Git stash命令详解:

应用场景：

1 当正在dev分支上开发某个项目，这时项目中出现一个bug，需要紧急修复，但是正在开发的内容只是完成一半，还不想提交，这时可以用git stash命令将修改的内容保存至堆栈区，然后顺利切换到hotfix分支进行bug修复，修复完成后，再次切回到dev分支，从堆栈中恢复刚刚保存的内容。

2 由于疏忽，本应该在dev分支开发的内容，却在master上进行了开发，需要重新切回到dev分支上进行开发，可以用git stash将内容保存至堆栈中，切回到dev分支后，再次恢复内容即可。

总的来说，git stash命令的作用就是将目前还不想提交的但是已经修改的内容进行保存至堆栈中，后续可以在某个分支上恢复出堆栈中的内容。这也就是说，stash中的内容不仅仅可以恢复到原先开发的分支，也可以恢复到其他任意指定的分支上。git stash作用的范围包括工作区和暂存区中的内容，也就是说没有提交的内容都会保存至堆栈中。

命令详解：

1 git stash

能够将所有未提交的修改（工作区和暂存区）保存至堆栈中，用于后续恢复当前工作目录。

$ git status

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: src/main/java/com/wy/CacheTest.java

modified: src/main/java/com/wy/StringTest.java

no changes added to commit (use "git add" and/or "git commit -a")

$ git stash

Saved working directory and index state WIP on master: b2f489c second

$ git status

On branch master

nothing to commit, working tree clean

2 git stash save

作用等同于git stash，区别是可以加一些注释，如下：

git stash的效果：

stash@{0}: WIP on master: b2f489c second

1

git stash save “test1”的效果：

stash@{0}: On master: test1

1

3 git stash list

查看当前stash中的内容

4 git stash pop

将当前stash中的内容弹出，并应用到当前分支对应的工作目录上。

注：该命令将堆栈中最近保存的内容删除（栈是先进后出）

顺序执行git stash save “test1”和git stash save “test2”命令，效果如下：

$ git stash list

stash@{0}: On master: test2

stash@{1}: On master: test1

$ git stash pop

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: src/main/java/com/wy/StringTest.java

no changes added to commit (use "git add" and/or "git commit -a")

Dropped refs/stash@{0} (afc530377eacd4e80552d7ab1dad7234edf0145d)

$ git stash list

stash@{0}: On master: test1

可见，test2的stash是首先pop出来的。

如果从stash中恢复的内容和当前目录中的内容发生了冲突，也就是说，恢复的内容和当前目录修改了同一行的数据，那么会提示报错，需要解决冲突，可以通过创建新的分支来解决冲突。

5 git stash apply

将堆栈中的内容应用到当前目录，不同于git stash pop，该命令不会将内容从堆栈中删除，也就说该命令能够将堆栈的内容多次应用到工作目录中，适应于多个分支的情况。

$ git stash apply

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: src/main/java/com/wy/StringTest.java

no changes added to commit (use "git add" and/or "git commit -a")

$ git stash list

stash@{0}: On master: test2

stash@{1}: On master: test1

堆栈中的内容并没有删除。

可以使用git stash apply + stash名字（如stash@{1}）指定恢复哪个stash到当前的工作目录。

6 git stash drop + 名称

从堆栈中移除某个指定的stash

7 git stash clear

清除堆栈中的所有 内容

8 git stash show

查看堆栈中最新保存的stash和当前目录的差异。

$ git stash show

src/main/java/com/wy/StringTest.java | 2 +-

1 file changed, 1 insertion(+), 1 deletion(-)

1

2

3

git stash show stash@{1}查看指定的stash和当前目录差异。

通过 git stash show -p 查看详细的不同：

$ git stash show -p

diff --git a/src/main/java/com/wy/CacheTest.java b/src/main/java/com/wy/CacheTest.java

index 6e90837..de0e47b 100644

--- a/src/main/java/com/wy/CacheTest.java

+++ b/src/main/java/com/wy/CacheTest.java

@@ -7,6 +7,6 @@ package com.wy;

\*/

public class CacheTest {

public static void main(String[] args) {

- System.out.println("git stash test");

+ System.out.println("git stash test1");

}

}

diff --git a/src/main/java/com/wy/StringTest.java b/src/main/java/com/wy/StringTest.java

index a7e146c..711d63f 100644

--- a/src/main/java/com/wy/StringTest.java

+++ b/src/main/java/com/wy/StringTest.java

@@ -12,7 +12,7 @@ public class StringTest {

@Test

public void test1() {

- System.out.println("=================");

+ System.out.println("git stash test1");

System.out.println(Strings.isNullOrEmpty(""));//true

System.out.println(Strings.isNullOrEmpty(" "));//false

System.out.println(Strings.nullToEmpty(null));//""

同样，通过git stash show stash@{1} -p查看指定的stash的差异内容。

9 git stash branch

从最新的stash创建分支。

应用场景：当储藏了部分工作，暂时不去理会，继续在当前分支进行开发，后续想将stash中的内容恢复到当前工作目录时，如果是针对同一个文件的修改（即便不是同行数据），那么可能会发生冲突，恢复失败，这里通过创建新的分支来解决。可以用于解决stash中的内容和当前目录的内容发生冲突的情景。

发生冲突时，需手动解决冲突。