git stash（git储藏）可用于以下情形：

发现有一个类是多余的，想删掉它又担心以后需要查看它的代码，想保存它但又不想增加一个脏的提交。这时就可以考虑git stash。

使用git的时候，我们往往使用分支（branch）解决任务切换问题，例如，我们往往会建一个自己的分支去修改和调试代码, 如果别人或者自己发现原有的分支上有个不得不修改的bug，我们往往会把完成一半的代码commit提交到本地仓库，然后切换分支去修改bug，改好之后再切换回来。这样的话往往log上会有大量不必要的记录。其实如果我们不想提交完成一半或者不完善的代码，但是却不得不去修改一个紧急Bug，那么使用git stash就可以将你当前未提交到本地（和服务器）的代码推入到Git的栈中，这时候你的工作区间和上一次提交的内容是完全一样的，所以你可以放心的修Bug，等到修完Bug，提交到服务器上后，再使用git stash apply将以前一半的工作应用回来。

经常有这样的事情发生，当你正在进行项目中某一部分的工作，里面的东西处于一个比较杂乱的状态，而你想转到其他分支上进行一些工作。问题是，你不想提交进行了一半的工作，否则以后你无法回到这个工作点。解决这个问题的办法就是git stash命令。储藏(stash)可以获取你工作目录的中间状态——也就是你修改过的被追踪的文件和暂存的变更——并将它保存到一个未完结变更的堆栈中，随时可以重新应用。

git stash用法

**1. stash保存当前修改**

git stash会把所有未提交的修改（包括暂存的和非暂存的）都保存起来，用于后续恢复当前工作目录。  
比如下面的中间状态，通过git stash命令推送一个新的储藏，当前的工作目录就干净了。

$ git status

On branch master

Changes to be committed:

new file: style.css

Changes not staged for commit:

modified: index.html

$ git stash

Saved working directory and index state WIP on master: 5002d47 our new homepage

HEAD is now at 5002d47 our new homepage

$ git status

On branch master

nothing to commit, working tree clean

需要说明一点，stash是本地的，不会通过git push命令上传到git server上。  
实际应用中推荐给每个stash加一个message，用于记录版本，使用git stash save取代git stash命令。示例如下：

$ git stash save "test-cmd-stash"

Saved working directory and index state On autoswitch: test-cmd-stash

HEAD 现在位于 296e8d4 remove unnecessary postion reset in onResume function

**$ git stash list (展示当前stash的情况)**

stash@{0}: On autoswitch: test-cmd-stash

2. 重新应用缓存的stash

可以通过git stash pop命令恢复之前缓存的工作目录，输出如下：

$ git status

On branch master

nothing to commit, working tree clean

$ git stash pop

On branch master

Changes to be committed:

new file: style.css

Changes not staged for commit:

modified: index.html

Dropped refs/stash@{0} (32b3aa1d185dfe6d57b3c3cc3b32cbf3e380cc6a)

这个指令将缓存堆栈中的第一个stash删除，并将对应修改应用到当前的工作目录下。  
你也可以使用git stash apply命令，将缓存堆栈中的stash多次应用到工作目录中，但并不删除stash拷贝。命令输出如下：

$ git stash apply

On branch master

Changes to be committed:

new file: style.css

Changes not staged for commit:

modified: index.html

3. 查看现有stash

可以使用git stash list命令，一个典型的输出如下：

$ git stash list

stash@{0}: WIP on master: 049d078 added the index file

stash@{1}: WIP on master: c264051 Revert "added file\_size"

stash@{2}: WIP on master: 21d80a5 added number to log

在使用git stash apply命令时可以通过名字指定使用哪个stash，默认使用最近的stash（即stash@{0}）。

4. 移除stash

可以使用git stash drop命令，后面可以跟着stash名字。下面是一个示例：

$ git stash list

stash@{0}: WIP on master: 049d078 added the index file

stash@{1}: WIP on master: c264051 Revert "added file\_size"

stash@{2}: WIP on master: 21d80a5 added number to log

$ git stash drop stash@{0}

Dropped stash@{0} (364e91f3f268f0900bc3ee613f9f733e82aaed43)

或者使用git stash clear命令，删除所有缓存的stash。

5. 查看指定stash的diff

可以使用git stash show命令，后面可以跟着stash名字。示例如下：

$ git stash show

index.html | 1 +

style.css | 3 +++

2 files changed, 4 insertions(+)

在该命令后面添加-p或--patch可以查看特定stash的全部diff，如下：

$ git stash show -p

diff --git a/style.css b/style.css

new file mode 100644

index 0000000..d92368b

--- /dev/null

+++ b/style.css

@@ -0,0 +1,3 @@

+\* {

+ text-decoration: blink;

+}

diff --git a/index.html b/index.html

index 9daeafb..ebdcbd2 100644

--- a/index.html

+++ b/index.html

@@ -1 +1,2 @@

+<link rel="stylesheet" href="style.css"/>

6. 从stash创建分支

如果你储藏了一些工作，暂时不去理会，然后继续在你储藏工作的分支上工作，你在重新应用工作时可能会碰到一些问题。如果尝试应用的变更是针对一个你那之后修改过的文件，你会碰到一个归并冲突并且必须去化解它。如果你想用更方便的方法来重新检验你储藏的变更，你可以运行 git stash branch，这会创建一个新的分支，检出你储藏工作时的所处的提交，重新应用你的工作，如果成功，将会丢弃储藏。

$ git stash branch testchanges

Switched to a new branch "testchanges"

# On branch testchanges

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#

# modified: index.html

#

# Changes not staged for commit:

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

#

# modified: lib/simplegit.rb

#

Dropped refs/stash@{0} (f0dfc4d5dc332d1cee34a634182e168c4efc3359)

这是一个很棒的捷径来恢复储藏的工作然后在新的分支上继续当时的工作。

7. 暂存未跟踪或忽略的文件

默认情况下，git stash会缓存下列文件：

添加到暂存区的修改（staged changes）

Git跟踪的但并未添加到暂存区的修改（unstaged changes）

但不会缓存一下文件：

在工作目录中新的文件（untracked files）

被忽略的文件（ignored files）

git stash命令提供了参数用于缓存上面两种类型的文件。使用-u或者--include-untracked可以stash untracked文件。使用-a或者--all命令可以stash当前目录下的所有修改。