`git stash` 命令用于将更改储藏在脏工作目录中。

使用语法

```
git stash list [<options>]
git stash show [<stash>]
git stash drop [-q|--quiet] [<stash>]
git stash ( pop | apply ) [--index] [-q|--quiet] [<stash>]
git stash branch <branchname> [<stash>]
git stash save [-p|--patch] [-k|--[no-]keep-index] [-q|--quiet]
         [-u|--include-untracked] [-a|--all] [<message>]
git stash [push [-p|--patch] [-k|--[no-]keep-index] [-q|--quiet]
         [-u|--include-untracked] [-a|--all] [-m|--message <message>]]
         [--] [<pathspec>…]]
git stash clear
git stash create [<message>]
git stash store [-m|--message <message>] [-q|--quiet] <commit>
```

描述

当要记录工作目录和索引的当前状态，但想要返回到干净的工作目录时，则使用 `git stash`。 该命令保存本地修改，并恢复工作目录以匹配 `HEAD` 提交。

这个命令所储藏的修改可以使用 `git stash list` 列出，使用 `git stash show` 进行检查，并使用 `git stash apply` 恢复(可能在不同的提交之上)。调用没有任何参数的 `git stash` 相当于 `git stash save`。 默认情况下，储藏列表为“分支名称上的 WIP”，但您可以在创建一个消息时在命令行上给出更具描述性的消息。

创建的最新储藏存储在 `refs/stash` 中; 这个引用的反垃圾邮件中会发现较旧的垃圾邮件，并且可以使用通常的 `reflog` 语法命名(例如，`stash@{0}`是最近创建的垃圾邮件，`stash@{1}`是 `stash@{2.hours.ago}`之前也是可能的)。也可以通过指定存储空间索引(例如整数 `n` 相当于储藏 `stash@{n}`)来引用锁存。

示例

以下是一些示例 -

1.拉取到一棵肮脏的树

当你处于某种状态的时候，你会发现有一些上游的变化可能与正在做的事情有关。当您的本地更改不会与上游的更改冲突时，简单的`git pull`将让您向前。

但是，有些情况下，本地更改与上游更改相冲突，`git pull`拒绝覆盖您的更改。 在这种情况下，您可以将更改隐藏起来，执行`git pull`，然后解压缩，如下所示：

```
 git pull
 ...
file foobar not up to date, cannot merge.
$ git stash
$ git pull
$ git stash pop
```

2.工作流中断

当你处于某种状态的时候，比如你的老板进来，要求立即开会或处理非常紧急的事务。 传统上，应该提交一个临时分支来存储您的更改，并返回到原始(`original`)分支进行紧急修复，如下所示：

```
# ... hack hack hack ...
$ git checkout -b my_wip
$ git commit -a -m "WIP"
$ git checkout master
$ edit emergency fix # 编辑内容
$ git commit -a -m "Fix in a hurry"
$ git checkout my_wip
$ git reset --soft HEAD^
# ... continue hacking ...
```

上面过程可以使用`git stash`来简化上述操作，如下所示：

```
# ... hack hack hack ...
$ git stash
$ edit emergency fix
$ git commit -a -m "Fix in a hurry"
$ git stash pop
# ... continue hacking ...
```

3.测试部分提交

当要从工作树中的更改中提交两个或多个提交时，可以使用`git stash save --keep-index`，并且要在提交之前测试每个更改：

```
# ... hack hack hack ...
$ git add --patch foo            # add just first part to the index
$ git stash save --keep-index    # save all other changes to the stash
$ edit/build/test first part
$ git commit -m 'First part'     # commit fully tested change
$ git stash pop                  # prepare to work on all other changes
# ... repeat above five steps until one commit remains ...
$ edit/build/test remaining parts
$ git commit foo -m 'Remaining parts'
```

4.恢复被错误地清除/丢弃的垃圾

如果你错误地删除或清除了垃圾，就不能通过正常的安全机制来恢复。 但是，您可以尝试以下命令来获取仍在存储库中但仍无法访问的隐藏列表：

```
git fsck --unreachable |
grep commit | cut -d\  -f3 |
xargs git log --merges --no-walk --grep=WIP
```

5.储藏你的工作

为了演示这一功能，可以进入你的项目，在一些文件上进行工作，有可能还暂存其中一个变更。如果运行`git status`，可以看到你的中间状态：

```
$ git status
# On branch master
# 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
```

```
#
```

现在你想要切换分支，但是还不想提交正在进行中的工作；所以储藏这些变更为了往堆栈推送一个新的储藏，只要运行 `git stash`：

```
$ git stash
Saved working directory and index state \
  "WIP on master: 049d078 added the index file"
HEAD is now at 049d078 added the index file
(To restore them type "git stash apply")
```

现在，工作目录就干净了：

```
$ git status
# On branch master
nothing to commit, working directory clean
```

这时，可以方便地切换到其他分支工作；变更都保存在栈上。要查看现有的储藏，可以使用 `git stash list`，如下所示 -

```
$ 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
```

在这个案例中，之前已经进行了两次储藏，所以你可以访问到三个不同的储藏。你可以重新应用你刚刚实施的储藏，所采用的命令就是之前在原始的 `stash` 命令的帮助输出里提示的：`git stash apply`。如果你想应用更早的储藏，可以通过名字指定它，像这样：`git stash apply stash@{2}`。如果不指明，Git 默认使用最近的储藏并尝试应用它：

```
$ git stash apply
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#
#       modified:   index.html
```

```
#       modified:   lib/simplegit.rb
#
```

对文件的变更被重新应用，但是被暂存的文件没有重新被暂存。想那样的话，必须在运行 `git stash apply` 命令时带上一个 `--index` 的选项来告诉命令重新应用被暂存的变更。如果是这么做的，应该已经回到原来的位置：

```
$ git stash apply --index
# On branch master
# 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
#
```

`apply` 选项只尝试应用储藏的工作——储藏的内容仍然在栈上。要移除它，可以运行 `git stash drop` 再加上希望移除的储藏的名字：

```
$ 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)
```

也可以运行 `git stash pop` 来重新应用储藏，同时立刻将其从堆栈中移走。

6.取消储藏

在某些情况下，可能想应用储藏的修改，在进行了一些其他的修改后，又要取消之前所应用储藏的修改。Git 没有提供类似于 `stash unapply` 的命令，但是可以通过取消该储藏的补丁达到同样的效果：

```
$ git stash show -p stash@{0} | git apply -R
```

同样的，如果沒有指定具体的某个储藏，Git 会选择最近的储藏：

```
$ git stash show -p | git apply -R
```

可能会想要新建一个别名，在你的 Git 里增加一个 `stash-unapply` 命令，这样更有效率。例如：

```
$ git config --global alias.stash-unapply '!git stash show -p | git apply -R'
$ git stash apply
$ #... work work work
$ git stash-unapply
```

7.从储藏中创建分支

如果储藏了一些工作，暂时不去理会，然后继续在你储藏工作的分支上工作，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在那之后修改过的文件，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储藏的变更，可以运行 `git stash branch`，这会创建一个新的分支，检出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)
```