

软件开发中，bug 就像家常便饭一样。有了 bug 就需要修复，在 Git 中，由于分支是如此的强大，所以，每个 bug 都可以通过一个新的临时分支来修复，修复后，合并分支，然后将临时分支删除。

当你接到一个修复一个代号 101 的 bug 的任务时，很自然地，你想创建一个分支 `issue-101` 来修复它，但是，等等，当前正在 `dev` 上进行的工作还没有提交：

```
$ git status

On branch dev

Changes to be committed:

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

    new file:   hello.py

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:   readme.txt
```

并不是你不想提交，而是工作只进行到一半，还没法提交，预计完成还需 1 天时间。但是，必须在两个小时内修复该 bug，怎么办？

幸好，Git 还提供了一个 `stash` 功能，可以把当前工作现场“储藏”起来，等以后恢复现场后继续工作：

```
$ git stash

Saved working directory and index state WIP on dev: f52c633 add merge
```

现在，用 `git status` 查看工作区，就是干净的（除非有没有被 Git 管理的文件），因此可以放心地创建分支来修复 bug。

首先确定要在哪个分支上修复 bug，假定需要在 `master` 分支上修复，就从 `master` 创建临时分支：

```
$ git checkout master

Switched to branch 'master'
```

```
Your branch is ahead of 'origin/master' by 6 commits.
```

```
(use "git push" to publish your local commits)
```

```
$ git checkout -b issue-101
```

```
Switched to a new branch 'issue-101'
```

现在修复 bug，需要把“Git is free software ...”改为“Git is a free software ...”，然后提交：

```
$ git add readme.txt
```

```
$ git commit -m "fix bug 101"
```

```
[issue-101 4c805e2] fix bug 101
```

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

修复完成后，切换到 `master` 分支，并完成合并，最后删除 `issue-101` 分支：

```
$ git checkout master
```

```
Switched to branch 'master'
```

```
Your branch is ahead of 'origin/master' by 6 commits.
```

```
(use "git push" to publish your local commits)
```

```
$ git merge --no-ff -m "merged bug fix 101" issue-101
```

```
Merge made by the 'recursive' strategy.
```

```
readme.txt | 2 +-  
1 file changed, 1 insertion(+), 1 deletion(-)
```

太棒了，原计划两个小时的 bug 修复只花了 5 分钟！现在，是时候接着回到 `dev` 分支干活了！

```
$ git checkout dev
```

```
Switched to branch 'dev'
```

```
$ git status

On branch dev

nothing to commit, working tree clean
```

工作区是干净的，刚才的工作现场存到哪去了？用 `git stash list` 命令看看：

```
$ git stash list

stash@{0}: WIP on dev: f52c633 add merge
```

工作现场还在，Git 把 `stash` 内容存在某个地方了，但是需要恢复一下，有两个办法：

一是用 `git stash apply` 恢复，但是恢复后，`stash` 内容并不删除，你需要用 `git stash drop` 来删除；

另一种方式是用 `git stash pop`，恢复的同时把 `stash` 内容也删了：

```
$ git stash pop

On branch dev

Changes to be committed:

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

    new file:   hello.py

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:   readme.txt

Dropped refs/stash@{0} (5d677e2ee266f39ea296182fb2354265b91b3b2a)
```

再用 `git stash list` 查看，就看不到任何 `stash` 内容了：

```
$ git stash list
```

你可以多次 `stash`，恢复的时候，先用 `git stash list` 查看，然后恢复指定的 `stash`，用命令：

```
$ git stash apply stash@{0}
```

小结

修复 `bug` 时，我们会通过创建新的 `bug` 分支进行修复，然后合并，最后删除；

当手头工作没有完成时，先把工作现场 `git stash` 一下，然后去修复 `bug`，修复后，再 `git stash pop`，回到工作现场。