1.1、在master（或其他分支）分支上，工作区不干净，如果切换到一个“未被”commit（跟踪）分支（新建且“没有被”commit过），是可以的；
1.2、在master（或其他分支）分支上，工作区不干净，如果切换到一个“被”commit（跟踪）分支（“已经被”commit过的），是不可以的；
2、master分支修改后不commit（工作区不干净），切到新建分支上（可以），在新分支上commit，而后回到master分支（之前的修改全部放弃了）
综上：1、master分支上新建新分支，则新分支和master同步（任何操作都会相互影响）；且可以相互切换（就算工作区不干净也可以切换）。2、一旦新分支被commit后，则这两个分支就还不再同步（相互之间更改不影响，除非合并）；且在“非干净分支”上不能切换！

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：

1、工作区不干净也可以创建、切换到新分支
2、master分支没提交，创建新BUG分支，修改的内容在BUG分支中也是同步的，当在BUG分支中修复问题后提交，master分支的修改就没了（工作区干净），内容变成上一次commit内容。
***因此为避免这种情况，需要提前将不需要提交的内容隐藏起来，然后再创建分支，在新的分支中修复bug，再提交；后合并，再将master隐藏的内容显示出来
3、第2项，条件是新创建的分支，且未commit。一旦commit，从master根本切换不到新分支

为啥需要隐藏？

```
$ 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？

1、如果在master分支上修改，需要在master上创建新分支，此时dev分支根本切换不到master分支（因为dev分支不干净）
2、如果直接在dev分支上创建新分支，虽然可以切换到新分支，但新分支提交后，dev分支的未commit工作全部放弃了。
所以：必须隐藏dev分支中未 提交的部分

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 switch 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 switch 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}
```

在master分支上修复了bug后，我们要想一想，dev分支是早期从master分支分出来的，所以，这个bug其实在当前dev分支上也存在。

那怎么在dev分支上修复同样的bug？重复操作一次，提交不就行了？

有木有更简单的方法？

有！

同样的bug，要在dev上修复，我们只需要把`4c805e2 fix bug 101`这个提交所做的修改“复制”到dev分支。注意：我们只想复制`4c805e2 fix bug 101`这个提交所做的修改，并不是把整个master分支merge过来。

为了方便操作，Git专门提供了一个`cherry-pick`命令，让我们能复制一个特定的提交到当前分支：

```
$ git branch
* dev
  master
$ git cherry-pick 4c805e2
[master 1d4b803] fix bug 101
 1 file changed, 1 insertion(+), 1 deletion(-)
```

分支1上修改bug0了，这个bug同样存在其他分支2中，在分支2上，只需要将分支1上修复的commit提交即可
也就是cherry-pick

Git自动给dev分支做了一次提交，注意这次提交的commit是`1d4b803`，它并不同于master的`4c805e2`，因为这两个commit只是改动相同，但确实是两个不同的commit。用`git cherry-pick`，我们就不需要在dev分支上手动再把修bug的过程重复一遍。

有些聪明的童鞋会想了，既然可以在master分支上修复bug后，在dev分支上可以“重放”这个修复过程，那么直接在dev分支上修复bug，然后在master分支上“重放”行不行？当然可以，不过你仍然需要`git stash`命令保存现场，才能从dev分支切换到master分支。

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；

在master分支上修复的bug，想要合并到当前dev分支，可以用`git cherry-pick <commit>`命令，把bug提交的修改“复制”到当前分支，避免重复劳动。

dev未完成工作不想commit，为啥要隐藏？
1、dev不commit，工作区不干净，就不能切换到master分支（master先前已经被commit）。
2、直接在dev创建新分支bug，虽然dev工作区不干净，但是仍然可以切换到bug（因为bug还没commit——相当于未被跟踪），当在bug分支上修复问题后commit，会把不想提交的未完成工作也提交了。dev分支上因为未提交，但是bug分支提交了，一旦bug分支commit就默认放弃dev分支的修改了，再merge bug分支就可以了，但是会把未完成工作也提交了。
****
综上：必须先隐藏，然后再创建bug分支，修复bug!
****