

第 1 个命令: `git status`

第 2 个命令: `git init`

第 3 个命令: `git add`

第 4 个命令: `git commit`

第5个命令 `git log`

总结

第6个命令 `git branch`

第7个命令: `git checkout`

第 8 个命令: `git merge`

第 9 个命令: `git branch -d` & `git branch -D`

第 10 个命令: `git tag`

先推荐一个不错的学习github的博客:

[https://blog.csdn.net/qq\\_35246620/article/details/66973794](https://blog.csdn.net/qq_35246620/article/details/66973794)

本文是基于上面博客里面的内容 进行学习 的;

使用git命令之前先在git里面输入如下命令:

```
1 请在命令行运行这五句话!!! 一定要运行这五句话, 不然 git 就不能用了
```

```
2
```

```
3 git config --global user.name 你的英文名字 #方便产品
```

```
4 git config --global user.email 你的常用邮箱 #方便产
```

```
5 git config --global push.default simple #
```

```
6 git config --global core.quotepath false #防
```

```
7 git config --global core.editor "vim"
```

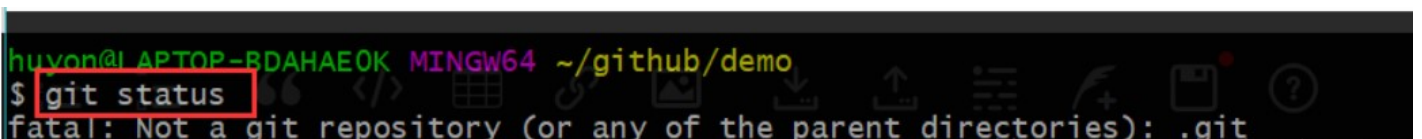


在 Git 中，所有的命令都是以git开头，例如，git init其作用就是初始一个 Git 仓库。  
为了方便演示，我们先在C盘的用户名目录下的github目录下创建一个名为demo的子目录,接下来我们的 Git 操作都是基于此目录和文件的;  
选择demo目录作为 Git 仓库，然后进入demo目录之中，点击鼠标右键，再选择Git Bash Here，即可打开 Git Bash 的命令行窗口：



## 第 1 个命令：git status

在命令行窗口的光标处，输入git status命令，查看仓库的状态:



如上图所示，结果显示demo不是一个 Git 仓库，这是很正常的反应，因为我们还没有在计算机中声明demo 为 Git 仓库;

## 第 2 个命令：git init

在命令行窗口的光标处，输入git init命令，初始化 Git 仓库：

```
huyon@LAPTOP-BDAHAEOK MINGW64 ~/github/demo
$ git init
Initialized empty Git repository in C:/Users/huyon/github/demo/.git/
```

如上图所示，结果显示已经初始化demo为一个空的 Git 仓库啦！

`touch hit.txt` 建立一个文本：

```
huyon@LAPTOP-BDAHAEOK MINGW64 ~/github/demo (master)
$ touch hit.txt
```

```
huyon@LAPTOP-BDAHAEOK MINGW64 ~/github/demo (master)
$ git status
On branch master
No commits yet
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        hit.txt
nothing added to commit but untracked files present (use "git add" to track)
```

如上图所示，在我们初始化仓库之后，demo目录已经成为一个 Git 仓库了，并且默认进入 Git 仓库的 master 分支，即主分支。在这里，我们需要注意的是 Untracked files 提示，它表示 demo 仓库中有文件没有被追踪，并提示了具体没有被追踪的文件为 hit.txt，还提示了我们可以使用 git add 命令操作这个文件。

## 第 3 个命令：git add

在命令行窗口的光标处，输入 `git add hit.txt` 命令，将 hit.txt 文件添加到 Git 仓库：

```
huyon@LAPTOP-BDAHAEOK MINGW64 ~/github/demo (master)
$ git add hit.txt
huyon@LAPTOP-BDAHAEOK MINGW64 ~/github/demo (master)
$ git status
On branch master
No commits yet
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   hit.txt
```

如上图所示，已经显示 Initial commit 初始化提交了，同时已经没有 Untracked files 提示了，这说明文件 hit.txt 已经被添加到 Git 仓库了，而在我们没有进行 git add 操作之前，文件 hit.txt 并不被 Git 仓库认可，因此才会出现提示初始化仓库为空的现象。在这里，需要声明一点，那就是：git add 命令并没有把文件提交到 Git 仓库，而是把文件添加到了「临时缓冲区」，这个命令有效防止了我们错误提交的可能性。

## 第 4 个命令：git commit

在命令行窗口的光标处，输入 `git commit -m "text commit"` 命令，将hit.txt文件提交到 Git 仓库：

```
@LAPTOP-BDAHAEOK MINGW64 ~/github/demo (master)
$ git commit -m "text commit"
[master (root-commit) 82d253f] text commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 hit.txt
```

如上图所示，我们成功将文件 `hit.txt` 提交到了 Git 仓库，其中 `commit` 表示提交，`-m` 表示提交信息，提交信息写在双引号"`"`内。接下来，再输入 `git status` 命令查看仓库状态：

```
@LAPTOP-BDAHAEOK MINGW64 ~/github/demo (master)
$ git status
On branch master
nothing to commit, working tree clean
```

如上图所示，结果显示 `nothing to commit, working tree clean`，这表示已经没有内容可以提交了，即全部内容已经提交完毕。

## 第5个命令 git log

在命令行窗口的光标处，输入`git log`命令，打印 Git 仓库提交日志：

```
@LAPTOP-BDAHAEOK MINGW64 ~/github/demo (master)
$ git log
commit 82d253f6998457146706fd4333c1bf6613f94f6c (HEAD -> master)
Author: richard <[redacted]@163.com>
Date: Wed Oct 3 10:06:35 2018 +0800
text commit
```

如上图所示，显示了我们的提交记录，提交记录的内容包括Author提交作者、Date提交日期和提交信息。

通过以上的操作，我们会发现一个现象，那就是：在每个git操作之后，我们基本都会输入`git status`命令，查看仓库状态。这也从侧面说明了`git status`命令使用的频率之高，也建议大家在操作 Git 仓库的时候多使用`git status`命令，这能帮助我们实时了解仓库的状态，显然非常有用。

## 总结

几个常见的命令：

第 1 个命令：`git status`：获取当前 仓库状态

第 2 个命令：`git init`：初始化仓库

第 3 个命令：`git add`：添加文件至仓库,`git add`命令并没有把文件提交到 Git 仓库，而是把文件添加到了「临时缓冲区」

第 4 个命令：`git commit`：提交文件至仓库

第5个命令: `git log`



## 第6个命令 git branch

```
huyon@LAPTOP-BDAHAEOK MINGW64 ~/github/demo (master)
$ git branch
* master
```

在命令行窗口的光标处，输入git branch命令，查看 Git 仓库的分支情况：

如上图所示，显示了仓库 demo 中的分支情况，现在仅有一个master分支，其中 master 分支前的 \* 号表示“当前所在的分支”，例如 \* master 就意味着我们所在的位置为demo仓库的主分支。输入命令git branch a，再输入命令git branch，结果如下图所示：

```
huyon@LAPTOP-BDAHAEOK MINGW64 ~/github/demo (master)
$ git branch a
huyon@LAPTOP-BDAHAEOK MINGW64 ~/github/demo (master)
$ git branch
a
* master
```

如上图所示，我们创建了一个名为a的分支，并且当前的位置仍然为主分支；

## 第7个命令： git checkout

在命令行窗口的光标处，输入 git checkout a 命令，切换到a分支：

```
huyon@LAPTOP-BDAHAEOK MINGW64 ~/github/demo (master)
$ git checkout a
Switched to branch 'a'
```

如上图所示，我们已经切换到 `a` 分支啦！也可以通过命令 `git branch` 查看分支情况：

```
huyon@LAPTOP-BDAHAEOK MINGW64 ~/github/demo (master)
$ git branch
* master

huyon@LAPTOP-BDAHAEOK MINGW64 ~/github/demo (master)
$ git branch a

huyon@LAPTOP-BDAHAEOK MINGW64 ~/github/demo (master)
$ git branch
a
* master

huyon@LAPTOP-BDAHAEOK MINGW64 ~/github/demo (master)
$ git checkout a
Switched to branch 'a'

huyon@LAPTOP-BDAHAEOK MINGW64 ~/github/demo (a)
$ git branch
* a
master
```

在这里，我们还有一个更简单的方法来查看当前的分支，即通过观察上图中用红色框圈起来的部分。此外，我们也可以在创建分支的同时，直接切换到新分支，命令为 `git checkout -b`，例如输入 `git checkout -b b` 命令：

```
huyon@LAPTOP-BDAHAEOK MINGW64 ~/github/demo (a)
$ git checkout -b b
Switched to a new branch 'b'
```

如上图所示，我们在a分支下创建b分支（b为a的分支），并直接切换到b分支。

## 第 8 个命令：git merge

切换到master分支，然后输入 `git merge a` 命令，将a分支合并到master分支：

```
huyon@LAPTOP-BDAHAEOK MINGW64 ~/github/demo (master)
$ git merge a
Already up-to-date.
```

如上图所示，我们已经将a分支合并到主分支啦！此外，在这里需要注意一点，那就是：在合并分支的时候，要考虑到两个分支是否有冲突，如果有冲突，则不能直接合并，需要先解决冲突；反之，则可以直接合并。

## 第 9 个命令：git branch -d & git branch -D

在命令行窗口的光标处，输入 `git branch -d a` 命令，删除a分支：

```
huyon@LAPTOP-BDAHAE0K MINGW64 ~/github/demo (master)
$ git branch -d a
Deleted branch a (was 82d253f).

huyon@LAPTOP-BDAHAE0K MINGW64 ~/github/demo (master)
$ git branch
b
* master
github/demo (master)
```

如上图所示，我们已经将分支a删除啦！不过有的时候，通过 `git branch -d` 命令可以出现删除不了现象，例如分支a的代码没有合并到主分支等，这时如果我们一定要删除该分支，那么我们可以通过命令 `git branch -D` 进行强制删除。

## 第 10 个命令：git tag

在命令行窗口的光标处，输入 `git tag v1.0` 命令，为当前分支添加标签：

```
huyon@LAPTOP-BDAHAE0K MINGW64 ~/github/demo (master)
$ git tag v1.0
fatal: tag 'v1.0' already exists

huyon@LAPTOP-BDAHAE0K MINGW64 ~/github/demo (master)
$ git tag
v1.0
```

如上图所示，我们为当前所在的a分支添加了一个v1.0标签。通过命令 `git tag` 即可查看标签记录：

```
huyon@LAPTOP-BDAHAE0K MINGW64 ~/github/demo (master)
$ git checkout v1.0
Note: checking out 'v1.0'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:
git checkout -b <new-branch-name>

HEAD is now at 82d253f... text commit
```

如上图所示，显示了我们添加标签的记录。通过命令 `git checkout v1.0` 即可切换到该标签下的代码状态：

通过「Git 初体验及其常用命令介绍」两篇博文的内容，我们已经了解了一些 Git 的常用命令啦，但还有很多命令我们没有进行演示，例如clone、rm、grep、pull和push等，Git 的魅力也并不止于此，还有更多精彩等待大家探索。