重新继续曾经未完成的工作

拉取代码

```
$ git clone git@github.com:ZebinGao/bluetooth.git
admin@zebin MINGW64 /d/BlueTestTry
$ ls
bluetooth/

admin@zebin MINGW64 /d/BlueTestTry
$ cd bluetooth

admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ ls
README.md

admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ git init
Reinitialized existing Git repository in D:/BlueTestTry/bluetooth/.git/
```

现在我们编写一个`readme.txt`文件，一定要放到`bluetooth`目录下（子目录也行），因为这是一个Git仓库

```
admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ ls
README.md  readme.txt
```

用命令`git add`告诉Git，把文件添加到仓库

```
admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ git add readme.txt
```

用命令`git commit`告诉Git，把文件提交到仓库

```
admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ git commit -m "wrote an new readme file"
[master 2557423] wrote an new readme file
 1 file changed, 2 insertions(+)
 create mode 100644 readme.txt
```

简单解释一下`git commit`命令，`-m`后面输入的是本次提交的说明，可以输入任意内容，当然最好是有意义的，这样你就能从历史记录里方便地找到改动记录。

嫌麻烦不想输入`-m "xxx"`行不行？确实有办法可以这么干，但是强烈不建议你这么干，因为输入说明对自己对别人阅读都很重要。实在不想输入说明的童鞋请自行 Google，我不告诉你这个参数。

`git commit` 命令执行成功后会告诉你，`1 file changed`：1 个文件被改动（我们新添加的 readme.txt 文件）；`2 insertions`：插入了两行内容（readme.txt 有两行内容）。

为什么Git添加文件需要`add`，`commit`一共两步呢？因为`commit`可以一次提交很多文件，所以你可以多次`add`不同的文件.

我们已经成功地添加并提交了一个readme.txt文件，现在，是时候继续工作了，于是，我们继续修改readme.txt文件.

现在，运行`git status`命令看看结果

```
admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   readme.txt

no changes added to commit (use "git add" and/or "git commit -a")


admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ git diff
diff --git a/readme.txt b/readme.txt
index 070bf74..c53dade 100644
--- a/readme.txt
+++ b/readme.txt
```

```
@@ -1,2 +1,2 @@
-Git is a version control system
+Git is a distributed version control system^M
 Git is free software
\ No newline at end of file
admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ git add readme.txt

admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   readme.txt
```

`git status`告诉我们，将要被提交的修改包括`readme.txt`，下一步，就可以放心地提交了

```
admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ git commit -m "add distributed"
[master 20c0b88] add distributed
 1 file changed, 1 insertion(+), 1 deletion(-)
```

`git log`命令显示从最近到最远的提交日志，我们可以看到 3 次提交，最近的一次是`append GPL`，上一次是`add distributed`，最早的一次是`wrote a readme file`。如果嫌输出信息太多，看得眼花缭乱的，可以试试加上`--pretty=oneline`参数：

```
admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ git log --pretty=oneline
dd1fcdda882fe8b95e519f9af2b39b9a8ea883be (HEAD -> master) append GPL
20c0b882831bf684dce1ef8579a82e063437e0d9 add distributed
2557423f9e1e3903eaf76283d761049cad8af4bf wrote an new readme file
```

好了，现在我们启动时光穿梭机，准备把`readme.txt`回退到上一个版本，也就是`add distributed`的那个版本，怎么做呢？

首先，Git 必须知道当前版本是哪个版本，在 Git 中，用 `HEAD` 表示当前版本，也就是最新的提交 `1094adb...`（注意我的提交 ID 和你的肯定不一样），上一个版本就是 `HEAD^`，上上一个版本就是 `HEAD^^`，当然往上 100 个版本写 100 个`^`比较容易数不过来，所以写成 `HEAD~100`。

现在，我们要把当前版本 `append GPL` 回退到上一个版本 `add distributed`，就可以使用 `git reset` 命令

```
admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ git reset --hard HEAD^
HEAD is now at 20c0b88 add distributed

admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ git log --pretty=oneline
20c0b882831bf684dce1ef8579a82e063437e0d9 (HEAD -> master) add distributed
2557423f9e1e3903eaf76283d761049cad8af4bf wrote an new readme file
```

`--hard` 参数有啥意义？`--hard` 会回退到上个版本的已提交状态，而`--soft` 会回退到上个版本的未提交状态，`--mixed` 会回退到上个版本已添加但未提交的状态。现在，先放心使用`--hard`。

```
admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ cat readme.txt
Git is a distributed version control system
Git is free software
```

最新的那个版本 `append GPL` 已经看不到了！好比你从 21 世纪坐时光穿梭机来到了 19 世纪，想再回去已经回不去了，肿么办？

办法其实还是有的，只要上面的命令行窗口还没有被关掉，你就可以顺着往上找啊找啊，找到那个 `append GPL` 的 `commit id` 是 `1094adb...`，于是就可以指定回到未来的某个版本：

```
admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ git reset --hard dd1fc
HEAD is now at dd1fcdd append GPL
```

在 Git 中，总是有后悔药可以吃的。当你用`$ git reset --hard HEAD^`回退到`add distributed`版本时，再想恢复到`append GPL`，就必须找到`append GPL`的 commit id。Git 提供了一个命令`git reflog`用来记录你的每一次命令

```
admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ git reflog
20c0b88 (HEAD -> master) HEAD@{0}: reset: moving to 20c0b
dd1fcdd HEAD@{1}: reset: moving to dd1fc
20c0b88 (HEAD -> master) HEAD@{2}: reset: moving to HEAD^
dd1fcdd HEAD@{3}: commit: append GPL
20c0b88 (HEAD -> master) HEAD@{4}: commit: add distributed
2557423 HEAD@{5}: commit: wrote an new readme file
d0abb08 (origin/master, origin/HEAD) HEAD@{6}: clone: from
git@github.com:ZebinGao/bluetooth.git
```

为什么 Git 比其他版本控制系统设计得优秀，因为 Git 跟踪并管理的是修改，而非文件。

你看，我们前面讲了，Git 管理的是修改，当你用`git add`命令后，在工作区的第一次修改被放入暂存区，准备提交，但是，在工作区的第二次修改并没有放入暂存区，所以，`git commit`只负责把暂存区的修改提交了，也就是第一次的修改被提交了，第二次的修改不会被提交。

提交后，用`git diff HEAD -- readme.txt`命令可以查看工作区和版本库里面最新版本的区别：

```
admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ git diff HEAD -- readme.txt
diff --git a/readme.txt b/readme.txt
```

```
index d7a4c3c..be13f15 100644
--- a/readme.txt
+++ b/readme.txt
@@ -1,4 +1,4 @@
 Git is a distributed version control system
 Git is free software distributed under the GPL
 Git has a mutable index called stage.
-Git tracks changes.
\ No newline at end of file
+Git tracks changes of files.
\ No newline at end of file
```

在 Git 中，删除也是一个修改操作，我们实战一下，先添加一个新文件 `test.txt` 到 Git 并且提交

```
admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ git add test.txt

admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ git commit -m "add test.txt"
[master 2ee2e28] add test.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 test.txt
```

一般情况下，你通常直接在文件管理器中把没用的文件删了，或者用 `rm` 命令删了：

```
admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ rm test.txt

admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ ls
LICENSE.txt  README.md  readme.txt

admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 7 commits.
  (use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
```

```
        deleted:    test.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

现在你有两个选择，一是确实要从版本库中删除该文件，那就用命令 `git rm` 删掉，并且 `git commit`

另一种情况是删错了，因为版本库里还有呢，所以可以很轻松地把误删的文件恢复到最新版本：

```
admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ git checkout -- test.txt

admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 7 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

首先，我们创建 `dev` 分支，然后切换到 `dev` 分支：

```
admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ git checkout -b dev
Switched to a new branch 'dev'

admin@zebin MINGW64 /d/BlueTestTry/bluetooth (dev)
$ git branch
* dev
  master
```

和远程仓库建立连接。

```
admin@zebin MINGW64 /d/BlueTestTry/bluetooth (dev)
$ git push
fatal: The current branch dev has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin dev



admin@zebin MINGW64 /d/BlueTestTry/bluetooth (dev)
$ git push --set-upstream origin dev
```

```
Total 0 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'dev' on GitHub by visiting:
remote:      https://github.com/ZebinGao/bluetooth/pull/new/dev
remote:
To github.com:ZebinGao/bluetooth.git
 * [new branch]      dev -> dev
Branch 'dev' set up to track remote branch 'dev' from 'origin'.
```

然后，我们就可以在 `dev` 分支上正常提交，比如对 `readme.txt` 做个修改，加上一行，然后提交。现在，`dev` 分支的工作完成，我们就可以切换回 `master` 分支

```
admin@zebin MINGW64 /d/BlueTestTry/bluetooth (dev)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$
```

`git merge` 命令用于合并指定分支到当前分支。合并后，再查看 `readme.txt` 的内容，就可以看到，和 `dev` 分支的最新提交是完全一样的。

注意到上面的 `Fast-forward` 信息，Git 告诉我们，这次合并是“快进模式”，也就是直接把 `master` 指向 `dev` 的当前提交，所以合并速度非常快。

当然，也不是每次合并都能 `Fast-forward`，我们后面会讲其他方式的合并。

合并完成后，就可以放心地删除 `dev` 分支了：

```
admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ git merge dev
Updating 2ee2e28..86b33dd
Fast-forward
 readme.txt | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ git branch -d dev
warning: not deleting branch 'dev' that is not yet merged to
         'refs/remotes/origin/dev', even though it is merged to HEAD.
error: The branch 'dev' is not fully merged.
If you are sure you want to delete it, run 'git branch -D dev'.

admin@zebin MINGW64 /d/BlueTestTry/bluetooth (master)
$ git branch -D dev
Deleted branch dev (was 86b33dd).
```