二、比较内容

下面将学习如何比较提交，分支等内容。

2.1 比较提交 - Git Diff

现在我们对项目做些修改：

```
$ cd gitproject
# 向README文件添加一行
$ echo "new line" >> README.md
# 添加新的文件file1
$ echo "new file" >> file1
```

使用 `git status` 查看当前修改的状态：

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

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.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)

	file1

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

可以看到一个文件修改了，另外一个文件添加了。如何查看修改的文件内容呢，那就需要使用 `git diff` 命令。`git diff` 命令的作用是比较修改的或提交的文件内容。

```
$ git diff
diff --git a/README.md b/README.md
index 21781dd..410e719 100644
--- a/README.md
+++ b/README.md
@@ -1, 2 +1, 3 @@
 gitproject
```

```
 ==========
+new line
```

上面的命令执行后需要使用 `q` 退出。命令输出当前工作目录中修改的内容，并不包含新加文件，请注意这些内容还没有添加到本地缓存区。

将修改内容添加到本地缓存区，通配符可以把当前目录下所有修改的新增的文件都自动添加：

```
$ git add *
```

再执行 `git diff` 会发现没有任何内容输出，说明当前目录的修改都被添加到了缓存区，如何查看缓存区内与上次提交之间的差别呢？需要使用 `--cached` 参数：

```
$ git diff --cached
diff --git a/README.md b/README.md
index 21781dd..410e719 100644
--- a/README.md
+++ b/README.md
@@ -1, 2 +1, 3 @@
 gitproject
 ==========
+new line
diff --git a/file1 b/file1
new file mode 100644
index 0000000..fa49b07
--- /dev/null
+++ b/file1
@@ -0, 0 +1 @@
+new file
```

可以看到输出中已经包含了新加文件的内容，因为 file1 已经添加到了缓存区。

最后我们提交代码：

```
$ git commit -m 'update code'
```

提交后 `git diff` 与 `git diff --cached` 都不会有任何输出了。

2.2 比较分支

可以用 git diff 来比较项目中任意两个分支的差异。

我们首先创建一个新的分支 `test`，并在该分支上提交一些修改：

```
# 创建test分支并切换到该分支
```

```
$ git branch test
$ git checkout test
# 添加新的一行到file1
$ echo "branch test" >> file1
# 创建新的文件file2
$ echo "new file2" >> file2
# 提交所有修改
$ git add *
 $ git commit -m 'update test branch'
```

然后，我们查看 test 分支和 master 之间的差别：

```
$ git diff master test
diff --git a/file1 b/file1
index fa49b07..17059cd 100644
--- a/file1
+++ b/file1
@@ -1 +1, 2 @@
 new file
+branch test
diff --git a/file2 b/file2
 new file mode 100644
 index 0000000..80e7991
 --- /dev/null
 +++ b/file2
 @@ -0, 0 +1 @@
 +new file2
```

`git diff` 是一个难以置信的有用的工具，可以找出你项目上任意两个提交点间的差异。可以使用 `git help diff` 详细查看其他参数和功能。

2.3 更多的比较选项

如果你要查看当前的工作目录与另外一个分支的差别，你可以用下面的命令执行:

```
# 切换到master
$ git checkout master

# 查看与test分支的区别
$ git diff test
diff --git a/file1 b/file1
index 17059cd..fa49b07 100644
--- a/file1
```

```
+++ b/file1
@@ -1, 2 +1 @@
 new file
-branch test
diff --git a/file2 b/file2
deleted file mode 100644
index 80e7991..0000000
--- a/file2
+++ /dev/null
@@ -1 +0, 0 @@
-new file2
```

你也以加上路径限定符，来只比较某一个文件或目录：

```
$ git diff test file1
diff --git a/file1 b/file1
index 17059cd..fa49b07 100644
--- a/file1
+++ b/file1
@@ -1, 2 +1 @@
 new file
-branch test
```

上面这条命令会显示你当前工作目录下的 file1 与 test 分支之间的差别。

`--stat` 参数可以统计一下有哪些文件被改动，有多少行被改动：

```
$ git diff test --stat
 file1 | 1 -
 file2 | 1 -
 2 files changed,  2 deletions（-）
```

三、分布式的工作流程

下面我们学习 Git 的分布式工作流程。

3.1 分布式的工作流程

你目前的项目在 `/home/shiyanlou/gitproject` 目录下，这是我们的 Git 仓库（repository），另一个用户也想与你协作开发。他的工作目录在这台机器上，如何让他提交代码到你的 Git 仓库呢?

首先，我们假设另一个用户也用 shiyanlou 用户登录，只是工作在不同的目录下开发代码，实际工作中不太可能发生，大部分情况都是多个用户，这个

假设只是为了让实验简化。

该用户需要从 Git 仓库进行克隆：

```
# 进入到临时目录
$ cd /tmp
# 克隆git仓库
$ git clone /home/shiyanlou/gitproject myrepo
$ ls -l myrepo
-rw-rw-r-- 1 shiyanlou shiyanlou 31 Dec 22 08:24 README.md
-rw-rw-r-- 1 shiyanlou shiyanlou 9 Dec 22 08:24 file1
```

这就建了一个新的 "myrepo" 的目录，这个目录里包含了一份gitproject仓库的克隆。这份克隆和原始的项目一模一样，并且拥有原始项目的历史记录。

在 myrepo 做了一些修改并且提交:

```
$ cd /tmp/myrepo

# 添加新的文件newfile
$ echo "newcontent" > newfile

# 提交修改
$ git add newfile
$ git commit -m "add newfile"
```

myrepo 修改完成后，如果我们想合并这份修改到 `gitproject` 的 git 仓库该如何做呢?

可以在仓库 `/home/shiyanlou/gitproject` 中把myrepo的修改给拉（pull）下来。执行下面几条命令:

```
$ cd /home/shiyanlou/gitproject
$ git pull /tmp/myrepo master
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From /tmp/myrepo
 * branch master -> FETCH_HEAD
Updating 8bb57aa..866c452
Fast-forward
 newfile | 1 +
 1 file changed, 1 insertion(+)
```

```
 create mode 100644 newfile

# 查看当前目录文件
$ ls
README.md file1 newfile
```

这就把 `myrepo` 的主分支合并到了 `gitproject` 的当前分支里了。

如果 `gitproject` 在 `myrepo` 修改文件内容的同时也做了修改的话，可能需要手工去修复冲突。

如果你要经常操作远程分支（remote branch），你可以定义它们的缩写:

```
$ git remote add myrepo /tmp/myrepo
```

git pull 命令等同于执行两个操作: 先使用 `git fetch` 从远程分支抓取最新的分支修改信息，然后使用 `git merge` 把修改合并进当前的分支。

`gitproject` 里可以用 `git fetch` 来执行 `git pull` 前半部分的工作， 但是这条命令并不会把抓下来的修改合并到当前分支里：

```
$ git fetch myrepo
From /tmp/myrepo
 * [new branch] master -> myrepo/master
```

获取后，我们可以通过 `git log` 查看远程分支做的所有修改，由于我们已经合并了所有修改，所以不会有任何输出：

```
$ git log -p master..myrepo/master
```

当检查完修改后，`gitproject` 可以把修改合并到它的主分支中：

```
$ git merge myrepo/master
Already up-to-date.
```

如果我们在 `myrepo` 目录下执行 `git pull` 会发生什么呢?

`myrepo` 会从克隆的位置拉取代码并更新本地仓库，就是把 `gitproject` 上的修改同步到本地:

```
# 进入到gitproject
$ cd /home/shiyanlou/gitproject

# 添加一行内容到newfile
$ echo "gitproject: new line" >> newfile

# 提交修改
$ git commit -a -m 'add newline to newfile'
[master 8c31532] add newline to newfile
```

```
 1 file changed,  1 insertion（+）

# 进入myrepo目录
$ cd /tmp/myrepo

# 同步gitproject的所有修改
$ git pull
remote: Counting objects: 6,  done.
remote: Compressing objects: 100%  （2/2）,  done.
remote: Total 3  （delta 1）,  reused 0  （delta 0）
Unpacking objects: 100%  （3/3）,  done.
From /home/shiyanlou/gitproject
 8bb57aa..8c31532 master -> origin/master
Updating 866c452..8c31532
Fast-forward
 newfile | 1 +
 1 file changed,  1 insertion（+）
```

因为 myrepo 是从 gitproject 仓库克隆的，那么他就不需要指定 gitproject 仓库的地 址。因为 Git 把 gitproject 仓库的地址存储到 myrepo 的配置文件中，这个地址就是在 git pull 时默认使用的远程仓库：

```
$ git config --get remote.origin.url
/home/shiyanlou/gitproject
```

如果 myrepo 和 gitproject 在不同的主机上，可以通过 ssh 协议来执行 clone 和pull 操作：

```
$ git clone localhost:/home/shiyanlou/gitproject test
```

这个命令会提示你输入 shiyanlou 用户的密码，用户密码随机，可以点击实验操作界面右侧工具栏的 SSH直连 按钮查看。

3.2 公共 Git 仓库

开发过程中，通常大家都会使用一个公共的仓库，并 clone 到自己的开发环境中，完成一个阶段的代码后可以告诉目标仓库的维护者来 pull 自己的代码。

如果你和维护者都在同一台机器上有帐号，那么你们可以互相从对 方的仓库目录里直接拉所作的修改，git 命令里的仓库地址也可以是本地的某个目录名：

```
$ git clone 仓库A的路径
$ git pull 仓库B的路径
```

也可以是一个ssh地址：

```
$ git clone ssh://服务器/账号/仓库名称
```

3.3 将修改推到一个公共仓库

通过 http 或是 git 协议，其它维护者可以通过远程访问的方式抓取（fetch）你最近的修改，但是他们没有写权限。如何将本地私有仓库的最近修改主动上传到公共仓库中呢?

最简单的办法就是用 `git push` 命令，推送本地的修改到远程 Git 仓库，执行下面的命令:

```
$ git push ssh://服务器仓库地址 master:master
```

或者

```
$ git push ssh://服务器仓库地址 master
```

`git push` 命令的目地仓库可以是 `ssh` 或 `http/https` 协议访问。

3.4 当推送代码失败时要怎么办

如果推送（push）结果不是快速向前 `fast forward`，可能会报像下面一样的错误：

```
error: remote 'refs/heads/master' is not an ancestor of
local 'refs/heads/master'.
Maybe you are not up-to-date and need to pull first?
error: failed to push to 'ssh://yourserver.com/~you/proj.git'
```

这种情况通常是因为没有使用 `git pull` 获取远端仓库的最新更新，在本地修改的同时，远端仓库已经变化了（其他协作者提交了代码），此时应该先使用 `git pull` 合并最新的修改后再执行 `git push`：

```
$ git pull
$ git push ssh://服务器仓库地址 master
```

四、Git标签

下面学习 Git 标签相关内容。

4.1 轻量级标签

我们可以用 git tag 不带任何参数创建一个标签（tag）指定某个提交（commit）：

```
# 进入到gitproject目录
$ cd /home/shiyanlou/gitproject

# 查看git提交记录
$ git log

# 选择其中一个记录标志位stable-1的标签，注意需要将后面的8c315325替换成仓库下的真实提交内，commit的名称很长，通常我们只需要写前面8位即可
$ git tag stable-1 8c315325

 # 查看当前所有tag
 $ git tag
 stable-1
```

这样，我们可以用stable-1 作为提交 `8c315325` 的代称。

前面这样创建的是一个“轻量级标签”。

如果你想为一个tag添加注释，或是为它添加一个签名， 那么我们就需要创建一个 "标签对象"。

标签对象

`git tag` 中使用 `-a`， `-s` 或是 `-u`三个参数中任意一个，都会创建一个标签对象，并且需要一个标签消息（tag message）来为 tag 添加注释。 如果没有 `-m` 或是 `-F` 这些参数，命令执行时会启动一个编辑器来让用户输入标签消息。

当这样的一条命令执行后，一个新的对象被添加到 Git 对象库中，并且标签引用就指向了一个标签对象，而不是指向一个提交，这就是与轻量级标签的区别。

下面是一个创建标签对象的例子:

```
$ git tag -a stable-2 8c315325 -m "stable 2"
$ git tag
stable-1
stable-2
```

4.2 签名的标签

签名标签可以让提交和标签更加完整可信。如果你配有`GPG key`，那么你就很容易创建签名的标签。首先你要在你的 `.git/config` 或 `~/.gitconfig` 里配好key。

下面是示例:

```
[user]
 signingkey = <gpg-key-id>
```

你也可以用命令行来配置:

```
$ git config （--global） user.signingkey <gpg-key-id>
```

现在你可以在创建标签的时候使用 `-s` 参数来创建“签名的标签”：

```
$ git tag -s stable-1 1b2e1d63ff
```

如果没有在配置文件中配 GPG key，你可以用 `-u` 参数直接指定。

```
$ git tag -u <gpg-key-id> stable-1 1b2e1d63ff
```

五、小结

本节学习了下面知识点：

- git diff
- 分布式的工作流程
- git tag