在使用 Git进行代码管理之前，我们首先要对 Git 进行初始化配置。

使用 Git 的第一件事就是设置你的名字和 `email`，这些就是你在提交 `commit` 时的签名，每次提交记录里都会包含这些信息。使用 `git config` 命令进行配置：

```
$ git config --global user.name "Scott Chacon"
$ git config --global user.email "schacon@gmail.com"
```

执行了上面的命令后，会在家目录（`/home/shiyanlou`）下建立一个叫 `.gitconfig` 的文件（该文件为隐藏文件，需要使用 `ls -al` 查看到）. 内容一般像下面这样，可以使用 vim 或 cat 查看文件内容:

```
$ cat ~/.gitconfig
[user]
 email = schacon@gmail.com
 name = Scott Chacon
```

上面的配置文件就是 Git 全局配置的文件，一般配置方法是 `git config --global <配置名称> <配置的值>`。

如果你想使项目里的某个值与前面的全局设置有区别（例如把私人邮箱地址改为工作邮箱），你可以在项目中使用 `git config` 命令不带 `--global` 选项来设置. 这会在你当前的项目目录下创建 `.git/config`，从而使用针对当前项目的配置。

三、获得一个Git仓库

既然我们现在把一切都设置好了，那么我们需要一个 Git 仓库。有两种方法可以得到它：一种是从已有的Git 仓库中 clone （克隆，复制）；还有一种是新建一个仓库，把未进行版本控制的文件进行版本控制。

3.1 Clone一个仓库

为了得到一个项目的拷贝（copy），我们需要知道这个项目仓库的地址（Git URL）。Git 能在许多协议下使用，所以 Git URL 可能以 `ssh://`, `http(s)://`, `git://` 开头。有些仓库可以通过多种协议来访问。

我们在 `github.com` 上提供了一个名字为 `gitproject` 的供大家测试的公有仓库，这个仓库可以使用下面方式进行 clone：

```
$ cd /home/shiyanlou/
$ git clone https://github.com/shiyanlou/gitproject
```

clone 操作完成后，会发现 `/home/shiyanlou` 目录下多了一个 `gitproject` 文件夹，这个文件夹里的内容就是我们刚刚 clone 下来的代码。由于当前 `gitproject` 仅是测试项目，里面仅有一个 README.md 文件。

```
$ cd gitproject/
 (master) $ ls
README.md
```

细心的同学可以发现在命令提示符 `$` 前面多了个 `(master)` 。这是由于实验楼的 Linux 使用的是zsh Shell，zsh 会判断当前的目录是否有 Git 仓库，如果是的话则自动把目前所在的 Git 分支显示在提示符中。Git 分支的概念我们会在稍后介绍。

3.2 初始化一个新的仓库

可以对一个已存在的文件夹用下面的命令让它置于 Git 的版本控制管理之下。

创建代码目录 `project`：

```
$ cd /home/shiyanlou/
$ mkdir project
```

进入到代码目录，创建并初始化Git仓库：

```
$ cd project
$ git init
```

Git 会输出:

```
Initialized empty Git repository in /home/shiyanlou/project/.git/
```

通过 `ls -la` 命令会发现project目录下会有一个名叫 `.git` 的目录被创建，这意味着一个仓库被初始化了。可以进入到 `.git` 目录查看下有哪些内容。

四、正常的工作流程

Git 的基本流程如下：

1. 创建或修改文件
2. 使用 `git add` 命令添加新创建或修改的文件到本地的缓存区（Index）
3. 使用 `git commit` 命令提交到本地代码库

4. （可选，有的时候并没有可以同步的远端代码库）使用`git push`命令将本地代码库同步到远端代码库

4.1 创建或修改文件

进入我们刚才建立的 `project` 目录，分别创建文件 `file1`，`file2`，`file3`：

```
$ cd /home/shiyanlou/project
$ touch file1 file2 file3
```

修改文件，可以使用 vim 编辑内容，也可以直接 echo 添加测试内容。

```
$ echo "test" >> file1
$ echo "test" >> file2
$ echo "test" >> file3
```

此时可以使用 `git status` 命令查看当前 Git 仓库的状态：

```
$ git status
On branch master

Initial commit

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

 file1
  file2
  file3
 nothing added to commit but untracked files present （use "git add" to tr
ack）
```

可以发现，有三个文件处于 `untracked` 状态，下一步我们就需要用 `git add` 命令将他们加入到缓存区（Index）。

4.2 使用 git add 加入缓存区

使用 `git add` 命令将新建的文件添加到缓存区：

```
$ git add file1 file2 file3
```

然后再次执行 `git status` 就会发现新的变化：

```
$ git status
On branch master
```

```
3
4 Initial commit
5
6 Changes to be committed:
7   (use "git rm --cached <file>..." to unstage)
8
9  new file: file1
10  new file: file2
11  new file: file3
```

你现在为 `commit` 做好了准备，你可以使用 `git diff` 命令再加上 `--cached` 参数，看看缓存区中哪些文件被修改了。进入到 `git diff --cached` 界面后需要输入 `q` 才可以退出：

```
1 $ git diff --cached
```

如果没有`--cached`参数，`git diff` 会显示当前你所有已做的但没有加入到缓存区里的修改。

如果你要做进一步的修改， 那就继续做， 做完后就把新修改的文件加入到缓存区中。

4.3 使用 git commit 提交修改

当所有新建，修改的文件都被添加到了缓存区，我们就要使用 `git commit` 提交到本地仓库：

```
1 $ git commit -m "add 3 files"
```

需要使用 `-m` 添加本次修改的注释，完成后就会记录一个新的项目版本。除了用 `git add` 命令，我们还可以用下面的 `-a` 参数将所有没有加到缓存区的修改也一起提交，但 `-a` 命令不会添加新建的文件。

```
1 $ git commit -a -m "add 3 files"
```

再次输入 `git status` 查看状态，会发现当前的代码库已经没有待提交的文件了，缓存区已经被清空。

至此，我们完成了第一次代码提交，这次提交的代码中我们创建了三个新文件。需要注意的是如果是修改文件，也需要使用 `git add` 命令添加到缓存区才可以提交。如果是删除文件，则直接使用 `git rm` 命令删除后会自动将已删除文件的信息添加到缓存区，`git commit` 提交后就会将本地仓库中的对应文件删除。

这时如果我们希望将本地仓库关联到远端服务器，我们可以使用 `git remote` 命令，不同于刚刚的 `git clone` 命令，直接将远端的仓库克隆下来。

我们当前的仓库是使用 `git init` 初始化的本地仓库，所以我们需要将本地仓库与远程仓库关联，使用如下命令（需要修改下面的远程仓库地址为自己的仓库地址）：

```
$ git remote add origin https://github.com/kinglion580/shiyanlou.git
```

对于上述命令而言，`git remote add` 命令用于添加远程主机，`origin` 是主机名，此处我们可以自定义，不一定非要使用 `origin`，

而 `https://github.com/kinglion580/shiyanlou.git`，是我自己的远程仓库，此处 需要替换为自己的远程仓库地址

这个时候如果本地的仓库连接到了远程Git服务器，可以使用下面的命令将本地仓库同步到远端服务器：

```
# 需要输入仓库对应的用户名和密码
$ git push origin master
```

五、分支与合并

Git 的分支可以让你在主线（master 分支）之外进行代码提交，同时又不会影响代码库主线。分支的作用体现在多人协作开发中，比如一个团队开发软件，你负责独立的一个功能需要一个月的时间来完成，你就可以创建一个分支，只把该功能的代码提交到这个分支，而其他同事仍然可以继续使用主线开发，你每天的提交不会对他们造成任何影响。当你完成功能后，测试通过再把你的功能分支合并到主线。

5.1 创建分支

一个 Git 仓库可以维护很多开发分支。现在我们来创建一个新的叫 experimental 的分支：

```
$ git branch experimental
```

运行 `git branch` 命令可以查看当前的分支列表，以及目前的开发环境处在哪个分支上：

```
$ git branch
  experimental
* master
```

5.2 切换分支

experimental 分支是你刚才创建的，master 分支是 Git 系统默认创建的主分支。星号标识了你当工作在哪个分支下，输入 `git checkout 分支名` 可以切换到其他分支：

```
$ git checkout experimental
Switched to branch 'experimental'
```

切换到 `experimental` 分支，切换完成后，先编辑里面的一个文件，再提交（commit）改动，最后切换回 master 分支：

```
# 修改文件file1
$ echo "update" >> file1
# 查看当前状态
$ git status
# 添加并提交file1的修改
$ git add file1
$ git commit -m "update file1"
# 查看file1的内容
$ cat file1
 test
 update
 # 切换到master分支
 $ git checkout master
```

查看下 `file1` 中的内容会发现刚才做的修改已经看不到了。因为刚才的修改时在 `experimental` 分支下，现在切换回了 `master` 分支，目录下的文件都是 `master` 分支上的文件了。

5.3 合并分支

现在可以在 `master` 分支下再作一些不同的修改：

```
# 修改文件file2
$ echo "update again" >> file2
# 查看当前状态
$ git status
# 添加并提交file2的修改
$ git add file2
$ git commit -m "update file2 on master"
# 查看file2的内容
$ cat file2
 test
 update again
```

这时，两个分支就有了各自不同的修改，分支的内容都已经不同，如何将多个分支进行合并呢?

可以通过下面的 `git merge` 命令来合并 `experimental` 到主线分支 `master`：

```
# 切换到master分支
$ git checkout master
# 将experimental分支合并到master
$ git merge -m 'merge experimental branch' experimental
```

`-m` 参数仍然是需要填写合并的注释信息。

由于两个 branch 修改了两个不同的文件，所以合并时不会有冲突，执行上面的命令后合并就完成了。

如果有冲突，比如两个分支都改了一个文件 file3，则合并时会失败。首先我们在master分支上修改file3 文件并提交：

```
# 切换到master分支
$ git checkout master
# 修改file3文件
$ echo "master: update file3" >> file3
# 提交到master分支
$ git commit -a -m 'update file3 on master'
```

然后切换到 experimental，修改 file3 并提交：

```
# 切换到experimental分支
$ git checkout experimental
# 修改file3文件
$ echo "experimental: update file3" >> file3
# 提交到experimental分支
$ git commit -a -m 'update file3 on experimental'
```

切换到 master 进行合并：

```
$ git checkout master
$ git merge experimental
Auto-merging file3
CONFLICT （content）: Merge conflict in file3
Automatic merge failed; fix conflicts and then commit the result.
```

合并失败后先用 `git status` 查看状态，会发现 file3 显示为 `both modified`，查看 file3内容会发现：

```
$ cat file3
test
<<<<<<< HEAD
master: update file3
=======
experimental: update file3
```

```
>>>>>>> experimental
```

上面的内容也可以使用 `git diff` 查看，先前已经提到 `git diff` 不加参数可以显示未提交到缓存区中的修改内容。

可以看到冲突的内容都被添加到了 file3 中，我们使用 vim 编辑这个文件，去掉 Git 自动产生标志冲突的 `<<<<<<` 等符号后，根据需要只保留我们需要的内容后保存，然后使用 `git add file3` 和 `git commit` 命令来提交合并后的 file3 内容，这个过程是手动解决冲突的流程。

```
# 编辑冲突文件
$ vim file3
# 提交修改后的文件
$ git add file3
$ git commit -m 'merge file3'
```

5.4 删除分支

当我们完成合并后，不再需要`experimental`时，可以使用下面的命令删除：

```
$ git branch -d experimental
```

`git branch -d`只能删除那些已经被当前分支的合并的分支. 如果你要强制删除某个分支的话就用`git branch -D`

5.5 撤销一个合并

如果你觉得你合并后的状态是一团乱麻，想把当前的修改都放弃，你可以用下面的命令回到合并之前的状态：

```
$ git reset --hard HEAD^
# 查看file3的内容，已经恢复到合并前的master上的文件内容
$ cat file3
```

5.6 快速向前合并

还有一种需要特殊对待的情况，在前面没有提到。通常，一个合并会产生一个合并提交（commit），把两个父分支里的每一行内容都合并进来。

但是，如果当前的分支和另一个分支没有内容上的差异，就是说当前分支的每一个提交（commit）都已经存在另一个分支里了，Git 就会执行一个 快速向前（fast forward）操作；Git 不创建任何新的提交（commit），只是将当前分支指向合并进来的分支。

六、Git 日志

下面我们来学习有关 Git 日志的内容。

6.1 查看日志

git log 命令可以显示所有的提交（commit）：

```
$ git log
```

如果提交的历史纪录很长，回车会逐步显示，输入 `q` 可以退出。

`git log` 有很多选项，可以使用 `git help log` 查看，例如下面的命令就是找出所有从 "v2.5 “ 开始在 fs 目录下的所有 Makefile 的修改（这个只是举例，不用操作）：

```
$ git log v2.5.. Makefile fs/
```

Git 会根据 git log 命令的参数，按时间顺序显示相关的提交（commit）。

6.2 日志统计

如果用 `--stat` 选项使用 `git log`，它会显示在每个提交（commit）中哪些文件被修改了， 这些文件分别添加或删除了多少行内容，这个命令相当于打印详细的提交记录：

```
$ git log --stat
```

6.3 格式化日志

你可以按你的要求来格式化日志输出。`--pretty` 参数可以使用若干表现格式，如 `oneline`：

```
$ git log --pretty=oneline
```

或者你也可以使用 `short` 格式：

```
$ git log --pretty=short
```

你也可用 `medium`， `full`， `fuller`， `email` 或 `raw`。 如果这些格式不完全符合你的需求， 你也可以用 `--pretty=format` 参数定义格式。

`--graph` 选项可以可视化你的提交图（commit graph），会用ASCII字符来画出一个很漂亮的提交历史（commit history）线：

```
$ git log --graph --pretty=oneline
```

6.4 日志排序

日志记录可以按不同的顺序来显示。如果你要指定一个特定的顺序，可以为 `git log` 命令添加顺序参数。

按默认情况，提交会按逆时间顺序显示，可以指定 `--topo-order` 参数，让提交按拓扑顺序来显示（就是子提交在它们的父提交前显示）：

```
$ git log --pretty=format:'%h : %s' --topo-order --graph
```

你也可以用 `--reverse` 参数来逆向显示所有提交日志。

七、小结

本节讲解了几个基本命令：

- git config：配置相关信息
- git clone：复制仓库
- git init：初始化仓库

- git add：添加更新内容到索引中
- git diff：比较内容
- git status：获取当前项目状况
- git commit：提交
- git branch：分支相关
- git checkout：切换分支
- git merge：合并分支
- git reset：恢复版本
- git log：查看日志