

二：Git 基本命令

- 创建版本库

版本库就是仓库，英文名为 repository,你可以把它理解成一个目录，这个目录里面的所有文件都可以被Git管理起来。

首先，建立一个目录并进入到这个目录中，运行 `git init`,这个目录就变成了一个git仓库。

```
mkdir test
cd test/
git init
```

```
limeng@revoserverx:~$ mkdir test
limeng@revoserverx:~$ cd test/
limeng@revoserverx:~/test$ pwd
/home/limeng/test
limeng@revoserverx:~/test$ git init
Initialized empty Git repository in /home/limeng/test/.git/
limeng@revoserverx:~/test$
```

我们发现它会提示如下，而且多了一个 `.git` 目录，这个目录是用来跟踪管理版本库的。

```
Initialized empty Git repository in /home/limeng/test/.git/
```

- 添加文件到版本库

第一步：我们创建一个文件并用 `git add` 命令把它添加到暂存区

```
touch a.txt
git add a.txt
git status
```

```
limeng@revoserverx:~/test$ touch a.txt
limeng@revoserverx:~/test$ git add a.txt
limeng@revoserverx:~/test$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   a.txt

limeng@revoserverx:~/test$
```

第二步：用git commit 命令把暂存区文件提交到仓库,使用git log 我们可以查看这条提交记录信息。

```
git commit -m "first commit"
```

```
limeng@revoserverx:~/test$ touch a.txt
limeng@revoserverx:~/test$ git add a.txt
limeng@revoserverx:~/test$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   a.txt

limeng@revoserverx:~/test$ git commit -m "first commit"
[master (root-commit) fc787af] first commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 a.txt
limeng@revoserverx:~/test$ git log
commit fc787af1ae6a29cc19a77566d6cdfda86085d36e
Author: limeng <limeng@revoview.com>
Date:   Mon May 28 16:46:41 2018 +0800

    first commit
limeng@revoserverx:~/test$
```

- 查看仓库状态 git status

本地和仓库是一致的显示 working directory clean

```
limeng@revoserverx:~/test$ git status
On branch master
nothing to commit, working directory clean
limeng@revoserverx:~/test$
```

修改 a.txt 文件后

```
limeng@revoserverx:~/test$ vi a.txt
limeng@revoserverx:~/test$ git status
On branch 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:   a.txt

no changes added to commit (use "git add" and/or "git commit -a")
limeng@revoserverx:~/test$
```

添加到缓存区 git add 之后

```
limeng@revoserverx:~/test$ git add a.txt
limeng@revoserverx:~/test$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   a.txt

limeng@revoserverx:~/test$
```

使用git diff 文件名查看文件的不同

```
diff --git a/a.txt b/a.txt
index e69de29..770cdf7 100644
--- a/a.txt
+++ b/a.txt
@@ -0,0 +1 @@
+this is my second commit
limeng@revoserverx:~/test$
```

使用 git difftool 查看文件的不同，会调出比对工具

```
This message is displayed because 'diff.tool' is not configured.
See 'git difftool --tool-help' or 'git help config' for more details.
'git difftool' will now attempt to use one of the following tools:
kompare emerge vimdiff

Viewing (1/1): 'a.txt'
Launch 'vimdiff' [Y/n]: y
2 files to edit
```

- **git 版本回退**

Git 一个commit就是一个快照，一旦出了问题可以进行回退。

显示提交记录 `git log`，可以看到刚才的提交记录

```

limeng@revoserverx:~/test$ git log
commit 22767fface89d969e28c3e42d12451e945de0896
Author: limeng <limeng@revoview.com>
Date: Mon May 28 17:17:29 2018 +0800

    modify a.txt

commit fc787af1ae6a29cc19a77566d6cdfda86085d36e
Author: limeng <limeng@revoview.com>
Date: Mon May 28 16:46:41 2018 +0800

    first commit
limeng@revoserverx:~/test$

```

把当前版本回退到上一个版本为 `git reset --hard HEAD^`

```

limeng@revoserverx:~/test$ git log --pretty=oneline
22767fface89d969e28c3e42d12451e945de0896 modify a.txt
fc787af1ae6a29cc19a77566d6cdfda86085d36e first commit
limeng@revoserverx:~/test$ git reset --hard HEAD^
HEAD is now at fc787af first commit
limeng@revoserverx:~/test$ git log
commit fc787af1ae6a29cc19a77566d6cdfda86085d36e
Author: limeng <limeng@revoview.com>
Date: Mon May 28 16:46:41 2018 +0800

    first commit
limeng@revoserverx:~/test$

```

如果想再回到未来的版本，窗口没有关闭的情况下，可以找到那个版本的id号，然后就可以回去 执行 `git reset --hard commit id号`

```

limeng@revoserverx:~/test$ git log --pretty=oneline
22767fface89d969e28c3e42d12451e945de0896 modify a.txt
fc787af1ae6a29cc19a77566d6cdfda86085d36e first commit
limeng@revoserverx:~/test$ git reset --hard HEAD^
HEAD is now at fc787af first commit
limeng@revoserverx:~/test$ git log
commit fc787af1ae6a29cc19a77566d6cdfda86085d36e
Author: limeng <limeng@revoview.com>
Date: Mon May 28 16:46:41 2018 +0800

    first commit
limeng@revoserverx:~/test$ git reset --hard 22767fface89d969e28c3e42d12451e945de0896
HEAD is now at 22767ff modify a.txt
limeng@revoserverx:~/test$ git log
commit 22767fface89d969e28c3e42d12451e945de0896
Author: limeng <limeng@revoview.com>
Date: Mon May 28 17:17:29 2018 +0800

    modify a.txt

commit fc787af1ae6a29cc19a77566d6cdfda86085d36e
Author: limeng <limeng@revoview.com>
Date: Mon May 28 16:46:41 2018 +0800

    first commit
limeng@revoserverx:~/test$

```

如果电脑关闭了，想恢复到新版本，找不到新版本的commit id怎么办呢？

Git提供一个命令可以用于记录你的每一次命令 `git reflog` 就可以在这找到新版本的commit id了

```

limeng@revoserverx:~/test$ git reflog
22767ff HEAD@{0}: reset: moving to 22767fface89d969e28c3e42d12451e945de0896
fc787af HEAD@{1}: reset: moving to HEAD^
22767ff HEAD@{2}: commit: modify a.txt
fc787af HEAD@{3}: reset: moving to fc787af1ae6a29cc19a77566d6cdfda86085d36e
ab4c779 HEAD@{4}: commit: modify a.txt
fc787af HEAD@{5}: commit (initial): first commit
limeng@revoserverx:~/test$

```

- 扩展, git log 的其他用法, [转自文章 git log 扩展用法](#)

`git log -p -2` -p 表示显示每次提交的差异内容 -2 表示显示最近的两次提交

```
limeng@revoserverx:~/test$ git log -p -2
commit 22767fface89d969e28c3e42d12451e945de0896
Author: limeng <limeng@revoview.com>
Date: Mon May 28 17:17:29 2018 +0800

    modify a.txt

diff --git a/a.txt b/a.txt
index e69de29..770cdf7 100644
--- a/a.txt
+++ b/a.txt
@@ -0,0 +1 @@
+this is my second commit

commit fc787af1ae6a29cc19a77566d6cdfda86085d36e
Author: limeng <limeng@revoview.com>
Date: Mon May 28 16:46:41 2018 +0800

    first commit

diff --git a/a.txt b/a.txt
new file mode 100644
index 0000000..e69de29
limeng@revoserverx:~/test$
```

`git log --stat` 显示简单的增改行统计

```
limeng@revoserverx:~/Android/Code/mocor_sc7731e01$ git log --stat
commit 30bf169877a5eaa78ae1ba3de1941ba9f0c53e74
Author: limeng <limeng@revoview.com>
Date: Fri May 25 15:14:19 2018 +0800

    Chuanqi-8803B removes gesture menu

    Change-Id: Iab2b783031741d1fb8036f7e69b8ce924f97dc70

 packages/apps/Settings/res_revo/values/config.xml | 20 -----
 packages/apps/Settings/src/com/android/settings/gestures/GesturesSettingPreferenceController.java | 24 ++++++++
 packages/apps/Settings/src/com/android/settings/search/SearchIndexableResources.java | 20 ++++++++
 zrevo/b55_xx08-chuanqi/env_b55_xx08-chuanqi.ini | 1 +
 zrevo/b55_xx08-chuanqi/overlay/8083B/packages/apps/Settings/res_revo/values/config.xml | 20 -----
 zrevo/features/default/env.ini | 8 ++++++
 zrevo/features/default/features.mk | 2 ++
 7 files changed, 33 insertions(+), 62 deletions(-)
```

如果嫌信息太多, 可以显示简要信息 `git log --pretty=oneline`

```
limeng@revoserverx:~/test$ git log --pretty=oneline
22767fface89d969e28c3e42d12451e945de0896 modify a.txt
fc787af1ae6a29cc19a77566d6cdfda86085d36e first commit
limeng@revoserverx:~/test$
```

`git log --pretty=short` 这个只显示commit信息

```
limeng@revoserverx:~/test$ git log --pretty=short
commit 22767fface89d969e28c3e42d12451e945de0896
Author: limeng <limeng@revoview.com>

    modify a.txt

commit fc787af1ae6a29cc19a77566d6cdfda86085d36e
Author: limeng <limeng@revoview.com>

    first commit
limeng@revoserverx:~/test$
```

除了上面的两个 pretty 还可以等于 full 或者 fuller

此外，最有用的是format，我们可以定制要显示的记录格式，像这样：

```
git log --pretty=format:"%h - %an, %ar : %s"

limeng@revoserverx:~/test$ git log --pretty=format:"%h - %an, %ar : %s"
22767ff - limeng, 46 minutes ago : modify a.txt
fc787af - limeng, 77 minutes ago : first commit
limeng@revoserverx:~/test$
```

下面列出了常用占位符写法及其意义

选项	说明
%H	提交对象（commit）的完整哈希字符串
%h	提交对象的简短哈希字符串
%T	树对象（tree）的完整哈希字符串
%t	树对象的简短哈希字符串
%P	父对象（parent）的完整哈希字符串
%p	父对象的简短哈希字符串
%an	作者（author）的名字
%ae	作者的电子邮件地址
%ad	作者修订日期（可以用 -date= 选项定制格式）
%ar	作者修订日期，按多久以前的方式显示
%cn	提交者(committer)的名字
%ce	提交者的电子邮件地址
%cd	提交日期
%cr	提交日期，按多久以前的方式显示
%s	提交说明

为什么会有作者和提交者呢，其实作者就是真正修改代码的人，而提交者是把修改提交到仓库的人。

用 oneline 或 format 时结合 --graph 选项，可以看到开头多出一些 ASCII 字符串表示的简单图形，形象地展示了每个提交所在的分支及其分化衍合情况

```
$ git log --pretty=format:"%h %s" --graph
* 2d3acf9 ignore errors from SIGCHLD on trap
* 5e3ee11 Merge branch 'master' of git://github.com/dustin/grit
|\
```

```
| * 420eac9 Added a method for getting the current branch.
* | 30e367c timeout code and tests
* | 5a09431 add timeout protection to grit
* | e1193f8 support for heads with slashes in them
|/
* d6016bc require time for xmlschema
* 11d191e Merge branch 'defunkt' into local
```

另外还有按照时间作限制的选项，比如 `--since` 和 `--until` 下面的命令列出所有最近两周内的提交：

```
$ git log --since=2.weeks
```

你可以给出各种时间格式，比如说具体的某一天（“2008-01-15”），或者是多久以前（“2 years 1 day 3 minutes ago”）。

还可以给出若干搜索条件，列出符合的提交。用 `--author` 选项显示指定作者的提交，用 `--grep` 选项搜索提交说明中的关键字。（请注意，如果要得到同时满足这两个选项搜索条件的提交，就必须用 `**--all-match` 选项**）

如果只关心某些文件或者目录的历史提交，可以在 `git log` 选项的最后指定它们的路径。因为是放在最后位置上的选项，所以用两个短划线（`--`）隔开之前的选项和后面限定的路径名。

```
git log --since="2018-1-1" -- ./packages/apps/Settings
```

查看尚未合并的变更

```
git log --no-merges master..
```

注意 `--no-merges` 标志意味着只显示没有合并到任何分支的变更，`master..` 选项，意思是指显示没有合并到 `master` 分支的变更（在 `master` 后面必须有 `..`）。

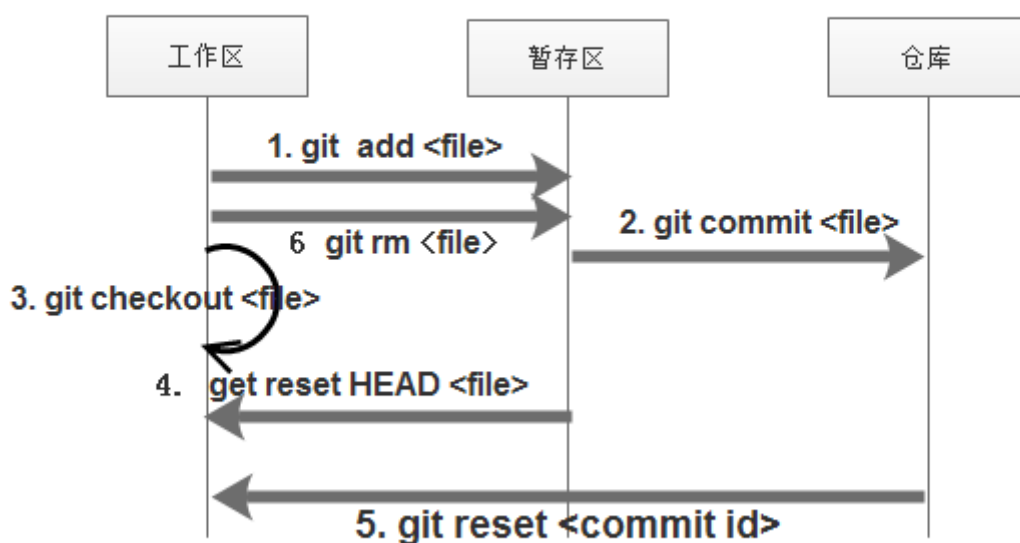
一些其他常用的选项及释义

选项	说明
<code>-p</code>	按补丁格式显示每个更新之间的差异。
<code>--stat</code>	显示每次更新的文件修改统计信息。
<code>--shortstat</code>	只显示 <code>--stat</code> 中最后的行数修改添加移除统计。
<code>--name-only</code>	仅在提交信息后显示已修改的文件清单。
<code>--name-status</code>	显示新增、修改、删除的文件清单。
<code>--abbrev-commit</code>	仅显示 SHA-1 的前几个字符，而非所有的 40 个字符。

选项	说明
--relative-date	使用较短的相对时间显示（比如，“2 weeks ago”）。
--graph	显示 ASCII 图形表示的分支合并历史。
--pretty	使用其他格式显示历史提交信息。可用的选项包括 oneline, short, full, fuller 和 format（后跟指定格式）。
-(n)	仅显示最近的 n 条提交
--since, --after	仅显示指定时间之后的提交。
--until, --before	仅显示指定时间之前的提交。
--author	仅显示指定作者相关的提交。
--committer	仅显示指定提交者相关的提交。
--no-merges	查看所有未被合并过的信息

- 检查当前文件状态

下面给出一个示意图，说明工作区缓存区和仓库



- git add命令把文件从工作区添加到暂存区。

- git commit提交更改，是把暂存区的内容提交到当前分支。
- 撤销修改
- 一种是工作区修改了，直接 `git checkout <file>` 回到和版本库一致。
- 另一种是已添加到暂存区的修改，撤销暂存区的修改：`git reset HEAD <file>` 回到了工作区修改状态。
- 如果从暂存区提交到了版本库，可以使用 `git reset <commit id>` 号回退,回到工作区修改状态，如果推送到了远程就没有办法了。
- 删除一个文件 `git rm <file>`
- 恢复删除的文件 `git reset HEAD <file> ()` `git checkout <file>`

创建远程仓库，注册 GitHub

注册 GitHub很简单，比较重要的是在github上添加我们的公钥，下面介绍如何在 windows 下使用 git 生成秘钥

windows 下生成秘钥

整个过程包含以下三步：

1. 设置 git 的 user name 和 email
2. 执行 `ssh-keygen -t rsa -C <email@example.com>`
3. 添加公钥到github

设置 user name 和 email

```
$ git config --global user.name "xxx"
$ git config --global user.email "xxxxxxxxx@163.com"
```

执行生成 ssh秘钥命令

```
$ ssh-keygen -t rsa -C "xxxxxxxxxxx@163.com" 按3个回车，密码为空。
```

最后在 C:/Users/\$USER/.ssh 目录下得到了两个文件：id_rsa和id_rsa.pub

添加秘钥到github

在github上添加 ssh 秘钥，需要添加的是 id_rsa.pub 公钥。在 Settings -> SSH and GPS Keys

Personal settings

Profile

Account

Emails

Notifications

Billing

SSH and GPG keys


Security

Blocked users

SSH keys


New SSH key

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

 **meng.li@mengli-PC**
Fingerprint: d5:5d:82:12:fc:3b:24:b2:70:25:57:54:ed:0e:91:8e
Added on Jun 17, 2016
Last used within the last 3 weeks — Read/write

SSH

Delete

 **home**
Fingerprint: 6a:63:6a:56:5b:bd:06:88:07:76:9b:ba:09:2b:9b:57
Added on Jun 28, 2017
Last used within the last week — Read/write

SSH

Delete

在github创建仓库并与本地关联

1.在github上创建一个仓库名为Test 2.关联本地仓库和远程仓库，在根目录下执行，这里需要把用户名修改为你自己的github用户名

```
$ git remote add origin git@github.com:<用户名>/Test.git
```

此时添加了一个远程仓库，我们查看 .git/config 文件发现多了 remote 这个配置

```
[core]
repositoryformatversion = 0
filemode = false
bare = false
logallrefupdates = true
symlinks = false
ignorecase = true
hideDotFiles = dotGitOnly
[remote "origin"]
url = git@github.com:BetterRamon/Test.git
fetch = +refs/heads/*:refs/remotes/origin/*
```

3. 将本地仓库推送到远程，-u 选项会指定一个默认主机，这样后面就可以不加任何参数使用 git push

```
$ git push -u origin master
```

此时我们再次查看 .git/config 文件

```
[core]
repositoryformatversion = 0
filemode = false
bare = false
logallrefupdates = true
symlinks = false
ignorecase = true
```

```
hideDotFiles = dotGitOnly
[remote "origin"]
url = git@github.com:BetterRamon/Test.git
fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
remote = origin
merge = refs/heads/master
```

此时测试一下，发现我们可以不带任何参数直接 git push

- 克隆代码

可以使用 SSH 或者 HTTP 两种方式克隆代码。

```
$ git clone git@github.com:<用户名>/mygit.git
$ git clone https://github.com/CoderMengLi/mygit.git
```

git 默认使用的是 SSH，使用 Http 比较慢，而且需要每次都输入口令。

- 忽略文件

通过创建 .gitignore 文件，可以忽略我们不想提交的文件。文件格式如下，支持通配符

```
*gen/
.settings/
*bin/
*.classpath
*.project
*project.properties
```

但是如果我们已经提交的文件，发现不需要，如 java bin目录下的 class文件，可以先清除被 trace 的文件，然后重新提交。

```
git rm -r --cached . //删除所有被 trace的文件
git add .
git commit -m 'add ignore'
```

有时候我们的文件被忽略了，添加不上，这时可以使用 -f 强制添加到 git

```
$ git add -f <file>
```

如果我们发现，可能是 .gitignore 写的有问题，导致文件被忽略，我们可以使用下列命令查找是那个配置忽略了文件。

```
$ git check-ignore -v xxx.class
.gitignore:3:*.class xxx.class
```

- **git diff 操作**

git diff 会使用文件补丁的格式显示具体添加和删除的行。

要查看尚未暂存的文件更新了哪些部分，直接输入 `git diff <file>` 此命令比较的是工作目录中当前文件和暂存区域快照之间的差异，也就是修改后还没暂存的变化内容。

```
limeng@revoserverx:~/test$ git diff b.txt
diff --git a/b.txt b/b.txt
index e69de29..f5b8f20 100644
--- a/b.txt
+++ b/b.txt
@@ -0,0 +1 @@
+this is diff test
limeng@revoserverx:~/test$
```

若要查看已暂存起来的文件和上次提交时的快照的差异，可以使用 `git diff --cached <file>`

```
limeng@revoserverx:~/test$ git diff --cached
diff --git a/b.txt b/b.txt
index f5b8f20..31bd8de 100644
--- a/b.txt
+++ b/b.txt
@@ -1 +1,2 @@
this is diff test
+git diff --cached
```

- **修改最后一次提交**

有时候我们提交完了发现某段代码写错了，或者提交信息有错误，想要撤销刚才的提交，可以使用 `--amend` 选项

```
$ git add a.txt
$ git commit --amend
```

上面的命令新添加了一个 `a.txt` 并且 `--amend` 命令可以修改我们最后一次提交的提交信息，`a.txt` 文件也会被加入到最后一次提交。

- **查看当前远程库**

要查看当前仓库有哪些远程库，可以使用 `git remote` 命令，它会列出每个远程库的简短名字，克隆某个项目后，至少可以看到一个名为 `origin` 的远程库。

```
$ git remote
origin
```

也可以加上 `-v` 选项，显示对应的克隆地址

```
$ git remote -v
origin git://github.com/xxx/xxx.git (fetch)
origin git://github.com/xxx/xxx.git (push)
```

- **添加远程仓库**

要添加一个远程仓库，可以为远程仓库指定一个简短的名字 `git remote add [shortname] [url]:`

如上面我们条件的github远程仓库

```
$ git remote add origin git@github.com:<用户名>/Test.git
```

- **从远程仓库抓取数据**

```
$ git fetch [remote-name]
```

此命令会到远程仓库中拉取所有你本地仓库中还没有的数据。运行该命令后你就可以获取到远程库中的所有分支，可以将其中某个分支合并到本地或者取出某个分支。

如果是克隆了一个仓库，远程仓库会自动归于 origin 名下。git fetch origin 会获取从你上次克隆以来别人上传到此远程仓库的所有更新，有一点很重要，fetch 命令只是将远端的数据拉到本地仓库，并不会自动合并到当前工作分支。

如果设置了某个分支用于跟踪某个远端仓库的分支（如何设置跟踪？），可以使用 git pull 命令自动抓取数据下来，然后将远端分支自动合并到本地仓库中当前分支。实际上，默认情况下 git clone 命令本质上就是自动创建了本地的 master 分支用于追踪远程仓库中的master分支（假设远程仓库有 master分支）。所以一般我们运行 git pull ,目的就是要从原始克隆的远端仓库中获取数据后，合并到工作目录中的当前分支。

- **推送数据到远程仓库**

将本地仓库中的数据推送到远程仓库，可以使用这个命令 `git push [remote-name] [branch-name]` 。

例如： 如果要把本地的master分支推送到origin服务器上（注意：克隆操作会自动使用默认的 master 和 origin 名字），运行如下命令：

```
$ git push origin master
```

- **查看远程仓库信息**

通过 `git remote show [remote-name]` 查看某个远程仓库的信息。

```
$ git remote show origin
```

- remote origin Fetch URL: review:mocor_sc7731eo1.git Push URL: review:mocor_sc7731eo1.git HEAD branch: master Remote branches: 360os tracked

master tracked official tracked Local branch configured for 'git pull': master merges with remote master Local ref configured for 'git push': master pushes to master (local out of date)

上面的信息高速我们如果是在 master 分支，就可以使用 git pull 命令抓取数据合并到本地。

- 远程仓库的删除和重命名

我们可以使用 git remote rename 命令修改某个远程仓库在本地的简称，如吧 struts 改为 pike

```
$ git remote
struts
$ git remote rename struts pike
$ git remote
pike
```

如果某个远程仓库不再使用，那么需要移除对用的远程仓库 使用 git remote rm 命令：

```
$ git remote
pike
$ git remote rm pike
$ git remote
$
```

- 设置本地分支和远程分支的对应关系

```
git branch --set-upstream-to=origin/<branch> master
```

- git 标签管理

发布一个版本时，通常先在版本库中打一个标签，这样，就唯一确定了打标签时刻的版本。下面介绍一些命令

操作	命令
打标签	git tag v1.0
可以在指定提交id处打标签	git tag v1.1 id号
显示tag信息	git show tag名
显示所有tag	git tag
创建带有说明的标签	git tag -a v1.2 -m "version1.2" id号

操作	命令
如果标签打错了, 也可以删除标签	git tag -d tag名
把标签推送到远程	git push origin v1.0
一次性推送	git push origin --tags
删除远程标签	① 先删除本地 git tag -d v0.1② 再删除远程, 删除语句是push git push origin :refs/tags/v0.1