

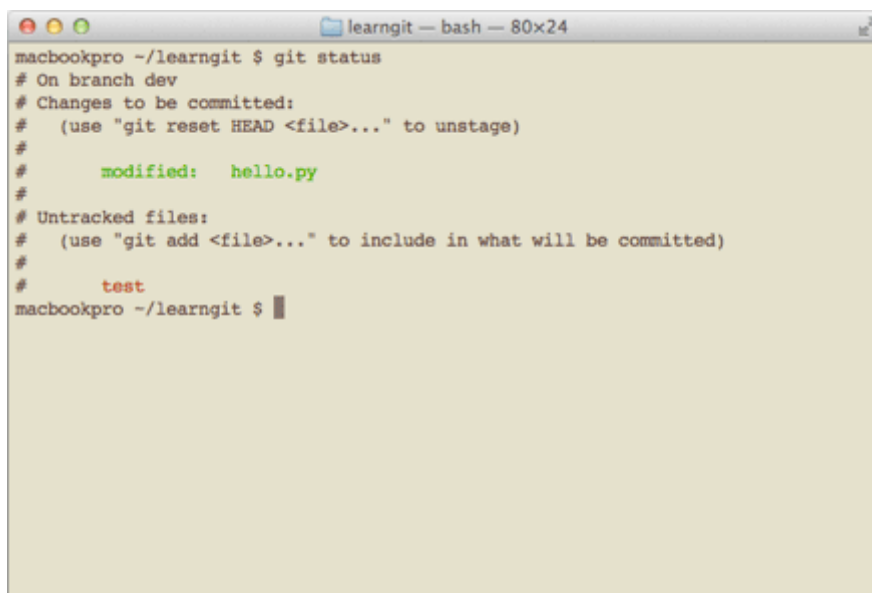
## 09 自定义Git

在[安装Git](#)一节中，我们已经配置了`user.name`和`user.email`，实际上，Git还有很多可配置项。

比如，让Git显示颜色，会让命令输出看起来更醒目：

```
$ git config --global color.ui true
```

这样，Git会适当地显示不同的颜色，比如`git status`命令：

A terminal window titled 'learnit - bash - 80x24' on a 'macbookpro' machine. The user has run the command 'git status'. The output shows the current branch is 'dev'. Under 'Changes to be committed', the file 'hello.py' is listed as 'modified' in green. Under 'Untracked files', the file 'test' is listed in red. The prompt returns to the user.

```
macbookpro ~/learnit $ git status
# On branch dev
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   hello.py
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       test
macbookpro ~/learnit $
```

文件名就会标上颜色。

我们在后面还会介绍如何更好地配置Git，以便让你的工作更高效。

### 忽略特殊文件

有些时候，你必须把某些文件放到Git工作目录中，但又不能提交它们，比如保存了数据库密码的配置文件啦，等等，每次`git status`都会显示`Untracked files ...`，有强迫症的童鞋心里肯定不爽。

好在Git考虑到了大家的感受，这个问题解决起来也很简单，在Git工作区的根目录下创建一个特殊的`.gitignore`文件，然后把要忽略的文件名填进去，Git就会自动忽略这些文件。

不需要从头写`.gitignore`文件，GitHub已经为我们准备了各种配置文件，只需要组合一下就可以使用了。所有配置文件可以直接在线浏览：<https://github.com/github/gitignore>

忽略文件的原则是：

1. 忽略操作系统自动生成的文件，比如缩略图等；

2. 忽略编译生成的中间文件、可执行文件等，也就是如果一个文件是通过另一个文件自动生成的，那自动生成的文件就没必要放进版本库，比如Java编译产生的`.class`文件；
3. 忽略你自己的带有敏感信息的配置文件，比如存放口令的配置文件。

举个例子：

假设你在Windows下进行Python开发，Windows会自动在有图片的目录下生成隐藏的缩略图文件，如果有自定义目录，目录下就会有`Desktop.ini`文件，因此你需要忽略Windows自动生成的垃圾文件：

```
# windows:
Thumbs.db
ehthumbs.db
Desktop.ini
```

然后，继续忽略Python编译产生的`.pyc`、`.pyo`、`dist`等文件或目录：

```
# Python:
*.py[cod]
*.so
*.egg
*.egg-info
dist
build
```

加上你自己定义的文件，最终得到一个完整的`.gitignore`文件，内容如下：

```
# windows:
Thumbs.db
ehthumbs.db
Desktop.ini

# Python:
*.py[cod]
*.so
*.egg
*.egg-info
dist
build

# My configurations:
db.ini
deploy_key_rsa
```

最后一步就是把`.gitignore`也提交到Git，就完成了！当然检验`.gitignore`的标准是`git status`命令是不是说`working directory clean`。

使用Windows的童鞋注意了，如果你在资源管理器里新建一个`.gitignore`文件，它会非常弱智地提示你必须输入文件名，但是在文本编辑器里“保存”或者“另存为”就可以把文件保存为`.gitignore`了。

有些时候，你想添加一个文件到Git，但发现添加不了，原因是这个文件被`.gitignore`忽略了：

```
$ git add App.class
The following paths are ignored by one of your .gitignore
files:
App.class
Use -f if you really want to add them.
```

如果你确实想添加该文件，可以用`-f`强制添加到Git：

```
$ git add -f App.class
```

或者你发现，可能是`.gitignore`写得有问题，需要找出来到底哪个规则写错了，可以用`git check-ignore`命令检查：

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

Git会告诉我们，`.gitignore`的第3行规则忽略了该文件，于是我们就可以知道应该修订哪个规则。

## 小结

- 忽略某些文件时，需要编写`.gitignore`；
- `.gitignore`文件本身要放到版本库里，并且可以对`.gitignore`做版本管理！

## 配置别名

有没有经常敲错命令？比如`git status`？`status`这个单词真心不好记。

如果敲`git st`就表示`git status`那就简单多了，当然这种偷懒的办法我们是极力赞成的。

我们只需要敲一行命令，告诉Git，以后`st`就表示`status`：

```
$ git config --global alias.st status
```

好了，现在敲`git st`看看效果。

当然还有别的命令可以简写，很多人都用`co`表示`checkout`，`ci`表示`commit`，`br`表示`branch`：

```
$ git config --global alias.co checkout
$ git config --global alias.ci commit
$ git config --global alias.br branch
```

以后提交就可以简写成：

```
$ git ci -m "bala bala bala..."
```

`--global` 参数是全局参数，也就是这些命令在这台电脑的所有Git仓库下都有用。

在[撤销修改](#)一节中，我们知道，命令 `git reset HEAD file` 可以把暂存区的修改撤销掉（`unstage`），重新放回工作区。既然是一个 `unstage` 操作，就可以配置一个 `unstage` 别名：

```
$ git config --global alias.unstage 'reset HEAD'
```

当你敲入命令：

```
$ git unstage test.py
```

实际上Git执行的是：

```
$ git reset HEAD test.py
```

配置一个 `git last`，让其显示最后一次提交信息：

```
$ git config --global alias.last 'log -1'
```

这样，用 `git last` 就能显示最近一次的提交：

```
$ git last
commit adca45d317e6d8a4b23f9811c3d7b7f0f180bfe2
Merge: bd6ae48 291bea8
Author: Michael Liao <askxuefeng@gmail.com>
Date: Thu Aug 22 22:49:22 2013 +0800

    merge & fix hello.py
```

甚至还有人丧心病狂地把 `lg` 配置成了：

```
git config --global alias.lg "log --color --graph --
pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s
%Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit"
```

来看看 `git lg` 的效果：

```
macbookpro ~/learngit $ git lg
* adca45d - (HEAD, origin/dev, dev) merge & fix hello.py (4 days ago) <Michael Liao>
|
| * 291bea8 - add /usr/bin/env (4 days ago) <Bob>
| * bd6ae48 - add coding: utf-8 (4 days ago) <Michael Liao>
|/
* fc38031 - add hello.py (4 days ago) <Michael Liao>
* 6224937 - add merge (4 days ago) <Michael Liao>
* 59bc1cb - conflict fixed (4 days ago) <Michael Liao>
|
| * 75a857c - AND simple (4 days ago) <Michael Liao>
| * 400b400 - & simple (4 days ago) <Michael Liao>
|/
* fec145a - (v0.2) branch test (4 days ago) <Michael Liao>
* d17efd8 - remove test.txt (6 days ago) <Michael Liao>
* 94cdc44 - add test.txt (6 days ago) <Michael Liao>
* 4378c15 - add changes of files (6 days ago) <Michael Liao>
* d4f25b6 - git tracks changes (6 days ago) <Michael Liao>
* 27c9860 - understand how stage works (6 days ago) <Michael Liao>
* 3628164 - append GPL (6 days ago) <Michael Liao>
* ea34578 - add distributed (6 days ago) <Michael Liao>
* cb926e7 - wrote a readme file (7 days ago) <Michael Liao>
macbookpro ~/learngit $
```

为什么不早点告诉我？别激动，咱不是为了多记几个英文单词嘛！

## 配置文件

配置Git的时候，加上`--global`是针对当前用户起作用的，如果不加，那只针对当前的仓库起作用。

配置文件放哪了？每个仓库的Git配置文件都放在`.git/config`文件中：

```
$ cat .git/config
[core]
    repositoryformatversion = 0
    filemode = true
    bare = false
    logallrefupdates = true
    ignorecase = true
    precomposeunicode = true
[remote "origin"]
    url = git@github.com:michaelliao/learngit.git
    fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
    remote = origin
    merge = refs/heads/master
[alias]
    last = log -1
```

别名就在`[alias]`后面，要删除别名，直接把对应的行删掉即可。

而当前用户的Git配置文件放在用户主目录下的一个隐藏文件`.gitconfig`中：

```
$ cat .gitconfig
[alias]
    co = checkout
    ci = commit
    br = branch
    st = status
[user]
    name = Your Name
    email = your@email.com
```

配置别名也可以直接修改这个文件，如果改错了，可以删掉文件重新通过命令配置。

## 小结

给Git配置好别名，就可以输入命令时偷个懒。我们鼓励偷懒。

## 搭建Git服务器

在[远程仓库](#)一节中，我们讲了远程仓库实际上和本地仓库没啥不同，纯粹为了7x24小时开机并交换大家的修改。

GitHub就是一个免费托管开源代码的远程仓库。但是对于某些视源代码如生命的商业公司来说，既不想公开源代码，又舍不得给GitHub交保护费，那就只能自己搭建一台Git服务器作为私有仓库使用。

搭建Git服务器需要准备一台运行Linux的机器，强烈推荐用Ubuntu或Debian，这样，通过几条简单的`apt`命令就可以完成安装。

假设你已经有`sudo`权限的用户账号，下面，正式开始安装。

第一步，安装`git`：

```
$ sudo apt-get install git
```

第二步，创建一个`git`用户，用来运行`git`服务：

```
$ sudo adduser git
```

第三步，创建证书登录：

收集所有需要登录的用户的公钥，就是他们自己的`id_rsa.pub`文件，把所有公钥导入到`/home/git/.ssh/authorized_keys`文件里，一行一个。

第四步，初始化Git仓库：

先选定一个目录作为Git仓库，假定是`/srv/sample.git`，在`/srv`目录下输入命令：

```
$ sudo git init --bare sample.git
```

Git就会创建一个裸仓库，裸仓库没有工作区，因为服务器上的Git仓库纯粹是为了共享，所以不让用户直接登录到服务器上去改工作区，并且服务器上的Git仓库通常都以`.git`结尾。然后，把owner改为git：

```
$ sudo chown -R git:git sample.git
```

第五步，禁用shell登录：

出于安全考虑，第二步创建的git用户不允许登录shell，这可以通过编辑`/etc/passwd`文件完成。找到类似下面的一行：

```
git:x:1001:1001:,,,:/home/git:/bin/bash
```

改为：

```
git:x:1001:1001:,,,:/home/git:/usr/bin/git-shell
```

这样，git用户可以正常通过ssh使用git，但无法登录shell，因为我们为git用户指定的`git-shell`每次一登录就自动退出。

第六步，克隆远程仓库：

现在，可以通过`git clone`命令克隆远程仓库了，在各自的电脑上运行：

```
$ git clone git@server:/srv/sample.git
Cloning into 'sample'...
warning: You appear to have cloned an empty repository.
```

剩下的推送就简单了。

## 管理公钥

如果团队很小，把每个人的公钥收集起来放到服务器的`/home/git/.ssh/authorized_keys`文件里就是可行的。如果团队有几百号人，就没法这么玩了，这时，可以用Gitosis来管理公钥。

这里我们不介绍怎么玩Gitosis了，几百号人的团队基本都在500强了，相信找个高水平的Linux管理员问题不大。

## 管理权限

有很多不但视源代码如生命，而且视员工为窃贼的公司，会在版本控制系统里设置一套完善的权限控制，每个人是否有读写权限会精确到每个分支甚至每个目录下。因为Git是为Linux源代码托管而开发的，所以Git也继承了开源社区的精神，不支持权限控制。不过，因为Git支持钩子（hook），所以，可以在服务器

端编写一系列脚本来控制提交等操作，达到权限控制的目的。[Gitolite](#)就是这个工具。

这里我们也不介绍[Gitolite](#)了，不要把有限的生命浪费到权限斗争中。

## 小结

- 搭建Git服务器非常简单，通常10分钟即可完成；
- 要方便管理公钥，用[Gito](#)；
- 要像SVN那样变态地控制权限，用[Gitolite](#)。