

git教程(6) 自定义git

一 忽略特殊文件

1 有时候，你必须把某些文件放到工作目录中，但又不能提交它们，如保存了数据库密码的配置文件等。

每次使用 `git status` 都会显示 `Untracked files ...` 【说的是该文件没有被 `git` 跟踪管理
!!!】

1.2 配置git不用跟踪管理的文件。通过 `.gitignore` 文件【可以使用正则表达式???!】，把要忽略的文件名填进去即可!!

2 .gitignore 忽略文件的原则

2.1 忽略操作系统自动生成的文件。如 缩略图

2.2 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 没必要提交到 版本库中。如 `java` 编译产生的 `.class`文件。

2.3 忽略自己的 带有敏感信息 的配置文件，如 存放口令的配置文件!!

3 举个例子。在 win下 进行 `python`的开发。

需要忽略 `Desktop.ini` 文件

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

然后继续忽略 `python` 编译产生的 `.pyc`、`.pyo`、`dist` 等文件或文件夹

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

注意：#windows、#python 说的是一下文件、目录 可能是 win、py 环境下产生的“不必要的东西、不必提交至版本库”！！

3.2 接着我们 最后再加上自己定义的文件【同时也不需要提交至版本库的】，这样就形成了完整的 .gitignore 文件：

```
# Windows:  
Thumbs.db  
ehthumbs.db  
Desktop.ini  
  
# Python:  
*.py[cod]  
*.so  
*.egg  
*.egg-info  
dist  
build  
  
# My configurations:  
db.ini  
deploy_key_rsa
```

3.3 最后一步就是把 .gitignore 文件也提交到 git 中。

当然检验 .gitignore 的标准是 git status 命令是不是提示 working directory 【工作目录，应该说的就是 工作区？！！】

4.1 写好了 .gitignore，后面有时有些文件就添加不进来了【因为可能该类型被写到了 .gitignore 中了！】！！

```
$ 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.
```

4.2 我们想强行【-f参数，force，强行添加进去！！】添加该文件也行：git add -f App.class

4.3 或者你发现可能是 .gitignore 写的有问题 ----> git check-ignore 。如 git check-ignore -v file_name

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

注意：上图说的是，.gitignore中的第3行规则忽略了该文件！！

5 小结

5.1 忽略某些文件【目录应该也行吧？？】或者 某类型文件时，需要编写 .gitignore 。

5.2 .gitignore文件本身【要提交至版本库里，被 git跟踪管理！！】要放到版本库里，并且可以对 .gitignore 做版本管理【即被 git跟踪、管理！！】

二 配置别名

1 有时 我们会把 git status 敲成 git stutas 等。那有没有别的办法让 status 进行简写呢？有，通过如 git config --global alias.st status

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

现在：git st 就等同于 git status 【盲猜，git应该是先自动把 st 替换成 status ,再去执行的！！】

1.1 配置 checkout、commit、branch 成 co、ci、br 【git config alias.新的缩写命令 旧的全写命令】

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

注意：--global是全局参数，也就是这些命令不仅仅在 当前版本库中可以使用，而是 在这台电脑的所有 git仓库下都有用！！

1.2 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 撤销unstage 操作，我们可以配置一个 unstage 的命令别名！！

如：`git config --global alias.unstage 'reset HEAD'`【因为是多个单词，它们之间有空格，需要加上 引号去包裹住，像之前是单个单词、就不必使用引号，直接 commit 等】

```
$ 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
```

1.3 配置仅显示最后一次提交的 缩写命令！！

`git config --global alias.last 'log -l'`

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

这样，用 `git last` 就能显示最近一次的提交：

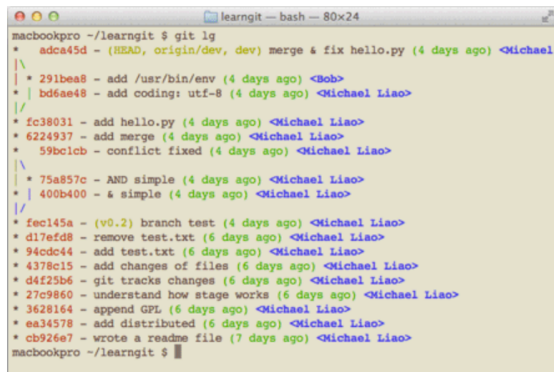
```
$ git last
commit adca45d317e6d8a4b23f9811c3d7b7f0f180bfe2
Merge: bd6ae48 291bea8
Author: Michael Liao <askxuefeng@gmail.com>
Date: Thu Aug 22 22:49:22 2013 +0800

merge & fix hello.py
```

还可以丧心病狂的将 `lg` 配置成：

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

来看看 `git lg` 的效果：



```
macbookpro ~/learnit $ git lg
* adca45d - (HEAD, origin/dev, dev) merge & fix hello.py (4 days ago) <Michael
|
| * 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>
* 59b61cb - 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 ~/learnit $
```

2 配置文件【--global是针对当前用户、当前一台电脑起作用，不加 --global 就是针对当前的仓库 生效而已！！】

2.1 配置 git 的时候加上 --global是针对当前用户、当前一台电脑起作用，不加 --global 就是针对当前的仓库 生效而已！！

2.2 每个仓库的配置文件放在了 .git/config【怎么感觉里面的配置内容有点像 XML啥的 格式???！！】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, 就直接把对应的行删除即可。

如 删掉 [alias]下面的 last = log -l 带有‘=’ 这一行！！

2.3 而当前用户、这台电脑的 git配置文件用在 用户主目录下【“故其配置后，生效范围比较大！！！”】 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
```

配置别名也可以在这里配置，若改错了或者不想要这个别名了，可以在该文件中去修改、删除！！

该文件的生效范围更大了，它是对“该用户、这台电脑”的所有版本库、git仓库生效！！

3 小结

3.1 给 git配置好别名，我们就可以在输入命令时偷懒了！！

3.2 添加 命令缩写别名，alias 是较核心的。如 `git config --global` 【对这台电脑的所有git仓库均生效！！】 `alias.last 'log -1'`。

三 搭建 git服务器

1 在远程仓库一节中，我们讲了远程仓库实际上与本地仓库没啥不同，纯粹为了 7 * 24小时开机并交换大家的修改。

1.1 github就是一个免费托管开源的远程仓库。但是对于某些不想开源 且 不想私人仓库而向GitHub交钱，那只能自己搭建一台 git服务器作为私有仓库的使用。

2 搭建git私人服务器的过程。【一台 Linux机器。推荐使用 Ubuntu、Debian。当然好像可以无脑的 gitlab！】

2.1 安装 git： `sudo apt-get install git`

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


2.2 创建一个git用户，用来运行git服务【? ? ?】

sodu adduser git

```
$ sudo adduser git
```

2.3 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，把所有公钥倒入 /home/git/.ssh/authorized_keys 文件里，一行一个。

2.4 初始化git仓库。

先选择一个目录作为git仓库，假定是 /srv/sample.git 。在 /srv 目录下输入命令： sudo git init --bare sample.git 【--bare 裸版本库? ?】

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

这时，git会创建一个 裸仓库【纯粹是为了共享!!! 无工作区!】 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区去改工作区。

并且服务器上的git仓库通常都是以 .git 结尾。然后，把 owner改为 git。sodu chown -R 【? ?】 git:git sample.git

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

2.5 为了安全，应禁用 shell登录 。---> 通过 类似 下面的一行。【原理? ? ? !】

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

改为：

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

这样，用户可以正常通过 ssh 使用 git,但无法使用 shell 登录!!

因为我们为 git用户指定的git-shell 每次一登录就自动退出!!!

2.6 克隆远程仓库【git clone git@server:/srv/sample.git】

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

3 管理公钥

3.1 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 文件里是可行的。如果团队有几百号人，就没法这么玩了，推荐使用 Gito 来管理公钥。

4 管理权限。

注意：git是为 linux源代码托管而开发的，所以 git 也继承了Linux社区的开源精神，故不支持 权限控制。

不过 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。达到控制权限的目的。Gitolite 就是这个工具！！

5 小结

5.1 用Linux【推荐使用 Ubuntu、Debian】搭建 git服务器比较简单

5.2 为了方便管理公钥，用 Gito【与下面权限管理的 Gitolite 是不一样的!!!】

5.3 想像SVN一样变态的控制权限，使用 Gitolite

整书完结