在windows上安装git

安装msysGit的项目提供了安装包：<http://msysgit.github.io/>

初次运行Git前的配置

配置个人用户名称

git config --global user.name “John Doe”

配置电子邮件地址

git config --global user.email joh@...

选项--global，指定了默认使用这里配置的用户信息

配置文本编辑器

git config --global core.editor emacs

默认使用vi或vim

配置差异分析工具

git config –global meerge.tool vimdiff

查看配置信息

git config –list

获取帮助

git help <verb>

git <verb> --help

man git-<verb>

Git基础

取得项目的Git仓库

从当前目录初始化

git init

对文件进行跟踪，然后提交

git add \*.c

git add README

git commit –m ‘initial project version’

从现有的仓库克隆

git clone [url]

git clone git://github.com/schacon/grit.git

这会在当前目录下创建一个名为“grit” 的目录，其中内含一个.git 的目录，

如果希望在克隆的时候，自己定义要新建的项目目录名称，可以在上面的命令最后指定：

git clone git://github.com/schacon/grit.git mygrit

记录每次更新到仓库

所有文件的状态：

已跟踪

已修改 Changed but not updated modified:

以放入暂存区 Changes to be committed

已提交

未跟踪 Untracked files

检查当前文件状态

git status

跟踪新文件

git add README

暂存更新

git add benchmarks.rb

忽略某些文件

创建一个名为.gitignore的文件，列出要忽略的文件模式

cat .gitignore

# 此为注释– 将被Git 忽略

\*.a # 忽略所有.a 结尾的文件

!lib.a # 但lib.a 除外

/TODO # 仅仅忽略项目根目录下的TODO 文件，不包括subdir/TODO

build/ # 忽略build/ 目录下的所有文件

doc/\*.txt # 会忽略doc/notes.txt 但不包括doc/server/arch.txt

查看已暂存和未暂存的更新

要查看未暂存的文件更新了那些部分

git diff

此命令比较的是工作目录中当前文件和暂存区域之间的差异，也就是修改之后还没有暂存起来的变化内容

git diff –cached or git diff –staged

提交更新

git commit

git commit –m “......”

跳过使用暂存区域

git commit –a

移除文件

从已跟踪文件清单中移除（是从暂存区域移除），然后提交

git rm 文件名

此命令连带从工作目录中删除指定文件，这样以后就不会出现在未跟踪文件清单中了

从工作目录中手工删除文件

rm 文件名

git rm 文件名

如果删除之前修改过并且已经存放到暂存区域，则必须用强制删除-f

把文件从Git仓库中删除（即从暂存区域移除），单任然希望保留在当前你工作目录中。即仅是从跟踪清单中删除，之后要在。gitignore文件中补上

git rm –cached 文件名

从面可以列出文件或者目录的名字，也可以使用glob模式

git rm log/\\*.log

删除所有log/目录下扩展名为.log的文件，其中\是Git自己的文件模式扩展匹配方式，此为递归删除

移动文件

git mv 原文件名 目标文件名

查看提交历史

git log

-p 展开显示每次提交的内容差异

-2 仅显示最近的两次更新

--stat 仅显示简要的增改行数统计

--pretty 指定使用完全不同于默认格式的方式展示提交历史

--pretty=oneline

将每个提交放在一行显示

--pretty=short

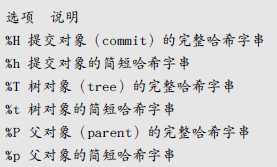
--pretty=full

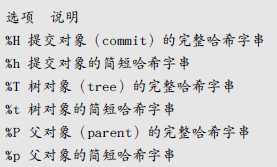
--pretty=fuller

--pretty=format

定制要显示的记录格式

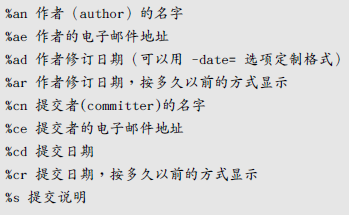
git log –pretty=format:”%h - %an, %ar : %s”





其实作者指的是实际作出修改的人，提交者指的是最后将此工作成果提交到仓库的人。

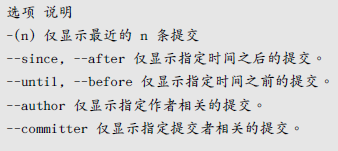
用oneline 或format 时结合--graph 选项，可以看到开头多出一些ASCII 字符串表示的简单图形，形象地展示了每个提交所在的分支及其分化衍合情况。

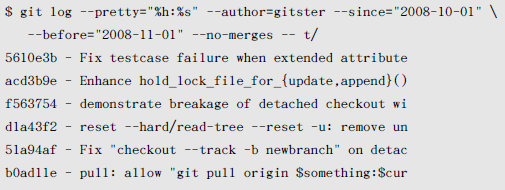


限制输出长度

列出所有最近两周内的提交

git log –since=2.weeks





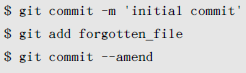
使用图形化工具查阅提交历史

gitk

撤销操作

修改最后一次提交

git commit –amend

上面的三条命令最终得到一个提交，第二个提交命令修正了第一个的提交内容。

取消已经暂存的文件

git reset HEAD 文件名

取消对文件的修改 （不可恢复）

git checkout – 文件名

如果只是想回退版本，同时保留刚才的修改以便将来继续工作，可以用下章介绍的stashing 和分支来处理，应该会更好

些。

远程仓库的使用

查看当前的远程仓库

git remote

在克隆完某个项目后，至少可以看到一个名为origin 的远程库，Git 默认使用这个名字来标识你所克隆的原始仓库

-v 显示对应的克隆地址（verbose）

添加远程仓库

git remote add [shortname] [url]

git remote add pb git://github.com/paulboone/ticgit.git

抓取所有Paul有的，但是本地仓库没有的信息，可以运行

git fetch pb

现在，Paul 的主干分支（master）已经完全可以在本地访问了，对应的名字是pb/master，你可以将它合并到自己的某个分支，或者切换到这个分支，看看有些什么有趣的更新。

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。运行完成后，你就可以在本地访问该远程仓库中的所有分支，将其中某个分支合并到本地，或者只是取出某个分支，一探究竟。

如果是克隆了一个仓库，此命令会自动将远程仓库归于origin 名下。所以，git fetchorigin 会抓取从你上次克隆以来别人上传到此远程仓库中的所有更新（或是上次fetch 以来别人提交的更新）。有一点很重要，需要记住，fetch 命令只是将远端的数据拉到本地仓库，并不自动合并到当前工作分支，只有当你确实准备好了，才能手工合并。

如果设置了某个分支用于跟踪某个远端仓库的分支（参见下节及第三章的内容），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。在日常工作中我们经常这么用，既快且好。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 分支用于跟踪远程仓库中的master 分支（假设远程仓库确实有master 分支）。所以一般我们运行git pull，目的都是要从原始克隆的远端仓库中抓取数据后，合并到工作目录中当前分支。

git pull

推送数据到远程仓库

git push [remote-name] [branch-name]

把本地的master分支推送到origin服务器上

git push origin master

只有在所克隆的服务器上有写权限，或者同一时刻没有其他人在推数据，这条命令才会如期完成任务。如果在你推数据前，已经有其他人推送了若干更新，那你的推送操作就会被驳回。你必须先把他们的更新抓取到本地，并到自己的项目中，然后才可以再次推送。

查看远程仓库信息

git remote show [remote-name]

它告诉我们，运行git push 时缺省推送的分支是什么（译注：最后两行）。它还显示了有哪些远端分支还没有同步到本地（译注：第六行的caching 分支），哪些已同步到本地的远端分支在远端服务器上已被删除（译注：Stale tracking branches 下面的两个分支），以及运行git pull 时将自动合并哪些分支（译注：前四行中列出的issues 和master 分支）。

远程仓库的删除和重命名

git remote rename 原名 修改后名

把pb改成paul

git remote rename pb paul

删除对应的远程仓库

git remote rm 库名

打标签

。。。见pro\_git\_中文版本.pdf 的P35

技巧和窍门

自动完成

如果你用的是Bash shell，可以试试看Git 提供的自动完成脚本。下载Git 的源代码，进入contrib/completion 目录，会看到一个git-completion.bash 文件。将此文件复制到你自己的用户主目录中（译注：按照下面的示例，还应改名加上点：cp gitcompletion.bash /.git-completion.bash），并把下面一行内容添加到你的.bashrc 文件中：

source ~/.git-completion.bash

Git分支

何为分支

（一次提交后仓库里的数据）

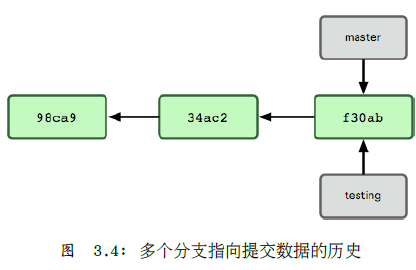
多次提交后的Git 对象数据

Git中的分支，本质就是指向commit对象的可变指针，Git会使用master作为分支的默认名字。 在若干次提交后，master分支指向最后一次提交对象。



创建一个新的分支

git branch testing



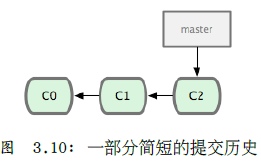
Git利用一个名为HEAD的特别指针指向当前正在工作的本地分支。而git branch命令仅是建立了一个新的分支，但不会自动切换到这个分支中，所以HEAD还是指向master

要切换到其他分支工作

git checkout testing

此时HEAD指向testing分支

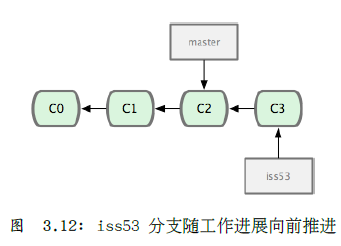
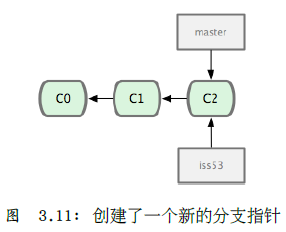
基本的分支与合并



git checkout –b iss53

等于： git branch iss53

git checkout iss53



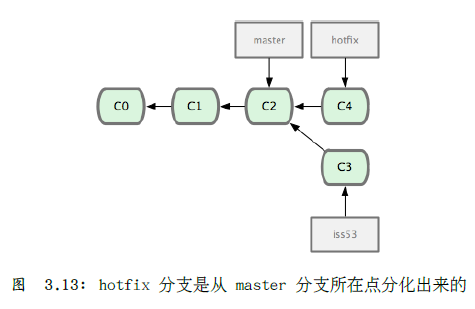
转换到master分支

git checkout master

转换分支时最好保持一个清洁的工作区域。

创建一个紧急修补分支hotfix

git checkout –b ‘hotfix’

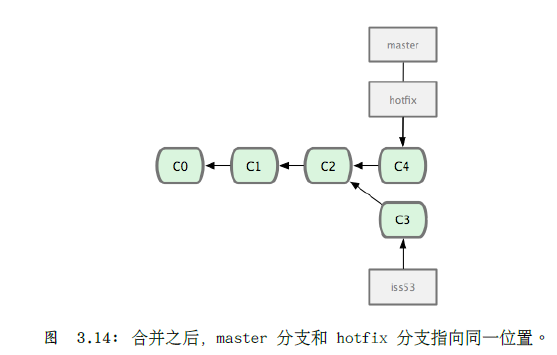


将紧急修补分支hotfix合并到master分支

git checkout master

git merge hotfix

合并时出现了“Fast forward”（快进）提示。由于当前master 分支所在的commit 是要并入的hotfix 分支的直接上游，Git 只需把指针直接右移。换句话说，如果顺着一个分支走下去可以到达另一个分支，那么Git 在合并两者时，只会简单地把指针前移，因为没有什么分歧需要解决，所以这个过程叫做快进（Fast forward）。

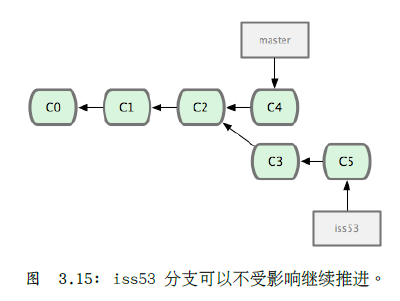


删除hotfix分支

git branch –d hotfix

回到问题53分支继续工作

git checkout iss53



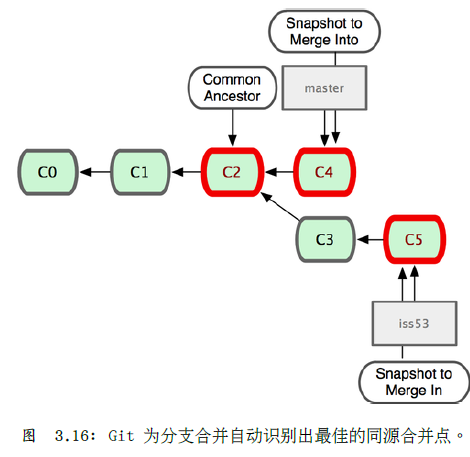
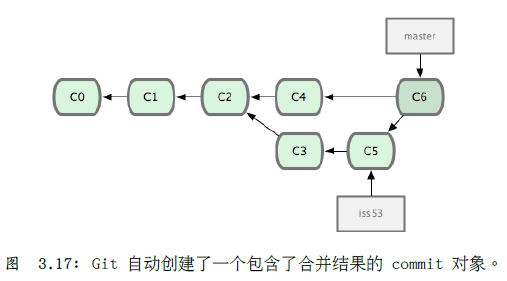
基本合并

问题53完成后，可以合并回master分支

git checkout master

git merge iss53

此次合并的开发历史从更早的地方开始分叉，master分支不是iss53分支的祖先。Git把三方合并的结果作一新的快照，并自动创建一个commit(c6)，这个c6叫合并提交

删除iss53

git branch –d iss53

冲突的合并

修改了两个待合并分支里的同一个文件的同一部分。

git merge iss53

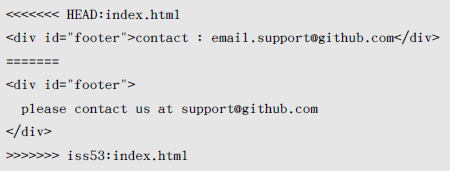
此时Git做了合并，但没有提交，他会等待你解决冲突

查看冲突

git status

任何包含未解决冲突的文件都会以未合并unmerged状态列出

Git 会在有冲突的文件里加入标准的冲突解决标记，可以通过它们来手工定位并解决这些冲突。



直接在文件中修改就可以解决冲突

运行git add将把它们标记为已解决resolved

也可以调用可视化合并工具

git mergetool

合并成功后就可以提交git commit

分支管理

列出所有分支

git branch

分支前的\*表示当前分支

查看各个分支最后一次commit信息

git branch –v

查看哪些分支已被合并入当前分支

git branch –merged

查看没有合并入当前分支的分支

git branch –no-merged

删除尚未合并的分支不能成功，只有用强制删除

git branch –D testing