Git详细教程

目录

[1 Git详细教程 1](#_Toc500165293)

[1.1 Git简介 1](#_Toc500165294)

[1.1.1 Git是何方神圣? 1](#_Toc500165295)

[1.1.2 重要的术语 1](#_Toc500165296)

[1.1.3 索引 1](#_Toc500165297)

[1.2 Git安装 1](#_Toc500165298)

[1.3 Git配置 1](#_Toc500165299)

[1.3.1 用户信息 1](#_Toc500165300)

[1.3.2 高亮显示 1](#_Toc500165301)

[1.3.3 忽略特定的文件 1](#_Toc500165302)

[1.3.4 使用.gitkeep来追踪空的文件夹 1](#_Toc500165303)

[1.4 开始操作Git 1](#_Toc500165304)

[1.4.1 创建内容 1](#_Toc500165305)

[1.4.2 创建仓库、添加文件和提交更改 1](#_Toc500165306)

[1.4.3 diff命令与commit更改 1](#_Toc500165307)

[1.4.4 Status, Diff 和 Commit Log 1](#_Toc500165308)

[1.4.5 更正提交的信息 - git amend 1](#_Toc500165309)

[1.4.6 删除文件 1](#_Toc500165310)

[1.5 远端仓库（remote repositories） 1](#_Toc500165311)

[1.5.1 设置一个远端的Git仓库 1](#_Toc500165312)

[1.5.2 推送更改到其他的仓库 1](#_Toc500165313)

[1.5.3 添加远端仓库 1](#_Toc500165314)

[1.5.4 显示已有的远端仓库 1](#_Toc500165315)

[1.5.5 克隆仓库 1](#_Toc500165316)

[1.5.6 拉取（Pull）更改 1](#_Toc500165317)

[1.5.7 还原更改 1](#_Toc500165318)

[1.5.8 标记 1](#_Toc500165319)

[1.6 分支、合并 1](#_Toc500165320)

[1.6.1 分支 1](#_Toc500165321)

[1.6.2 合并 1](#_Toc500165322)

[1.6.3 删除分支 1](#_Toc500165323)

[1.6.4 推送（push）一个分支到远端仓库 1](#_Toc500165324)

[1.7 解决合并冲突 1](#_Toc500165325)

[1.8 变基（Rebase） 1](#_Toc500165326)

[1.8.1 在同一分支中应用Rebase Commit 1](#_Toc500165327)

[1.8.2 Rebasing多个分支 1](#_Toc500165328)

[1.8.3 Rebase最佳实践 1](#_Toc500165329)

[1.8.4 创建和应用补丁 1](#_Toc500165330)

[1.9 定义同名命令 1](#_Toc500165331)

[1.10 放弃跟踪文件 1](#_Toc500165332)

[1.11 其他有用的命令 1](#_Toc500165333)

[1.12 安装Git服务 1](#_Toc500165334)

[1.13 在线的远端仓库 1](#_Toc500165335)

[1.13.1 克隆远端仓库 1](#_Toc500165336)

[1.13.2 添加远端仓库 1](#_Toc500165337)

[1.13.3 通过http和代理服务器进行远端操作 1](#_Toc500165338)

[1.14 Git服务提供商 1](#_Toc500165339)

[1.14.1 GitHub 1](#_Toc500165340)

[1.14.2 Bitbucket 1](#_Toc500165341)

[1.15 Git的图形接口 1](#_Toc500165342)

[1.16 Kindle版本教程 1](#_Toc500165343)

[1.17 问题与讨论 1](#_Toc500165344)

[1.18 链接和文章 1](#_Toc500165345)

# 1 Git详细教程

## 1.1 Git简介

### 1.1.1 Git是何方神圣?

Git是用C语言开发的分布版本控制系统。版本控制系统可以保留一个文件集合的历史记录，并能回滚文件集合到另一个状态（历史记录状态）。另一个状态可以是不同的文件，也可以是不同的文件内容。举个例子，你可以将文件集合转换到两天之前的状态，或者你可以在生产代码和实验性质的代码之间进行切换。文件集合往往被称作是“源代码”。在一个分布版本控制系统中，每个人都有一份完整的源代码（包括源代码所有的历史记录信息），而且可以对这个本地的数据进行操作。分布版本控制系统不需要一个集中式的代码仓库。

当你对本地的源代码进行了修改，你可以标注他们跟下一个版本相关（将他们加到index中），然后提交到仓库中来（commit）。Git保存了所有的版本信息，所以你可以转换你的源代码到任何的历史版本。你可以对本地的仓库进行代码的提交，然后与其他的仓库进行同步。你可以使用Git来进行仓库的克隆（clone）操作，完整的复制一个已有的仓库。仓库的所有者可以通过push操作（推送变更到别处的仓库）或者Pull操作（从别处的仓库拉取变更）来同步变更。

Git支持分支功能（branch）。如果你想开发一个新的产品功能，你可以建立一个分支，对这个分支的进行修改，而不至于会影响到主支上的代码。

Git提供了命令行工具；这个教程会使用命令行。你也可以找到图形工具，譬如与Eclipse配套的EGit工具，但是这些都不会在这个教程中进行描述。

### 1.1.2 重要的术语

Git 术语

| **术语** | **定义** |
| --- | --- |
| 仓库 | 一个仓库包括了所有的版本信息、所有的分支和标记信息. |
| Repository | 在Git中仓库的每份拷贝都是完整的。仓库让你可以从中 |
|  | 取得你的工作副本。 |
|  | 一个分支意味着一个独立的、拥有自己历史信息的代码线 |
| 分支 | （code line）。你可以从已有的代码中生成一个新的分支 |
| Branches | ，这个分支与剩余的分支完全独立。默认的分支往往是叫 |
|  | master。用户可以选择一个分支，选择一个分支叫做 |
|  | checkout. |
| 标记 | 一个标记指的是某个分支某个特定时间点的状态。通过标 |
| Tags | 记，可以很方便的切换到标记时的状态，例如2009年1月25 |
|  | 号在testing分支上的代码状态 |
| 提交 | 提交代码后，仓库会创建一个新的版本。这个版本可以在 |
| Commit | 后续被重新获得。每次提交都包括作者和提交者，作者和 |
|  | 提交者可以是不同的人 |
| URL | URl用来标识一个仓库的位置 |
|  | 用来表示代码的一个版本状态。Git通过用SHA1 hash算法 |
| 修订 | 表示的id来标识不同的版本。每一个 SHA1 id都是160位长 |
| Revision | ,16进制标识的字符串.最新的版本可以通过HEAD来获取. |
|  | 之前的版本可以通过"HEAD~1"来获取，以此类推。 |

### 1.1.3 索引

Git 需要将代码的变化显示的与下一次提交进行关联。举个例子，如果你对一个文件继续了修改，然后想将这些修改提交到下一次提交中，你必须将这个文件提交到索引中，通过git add file命令。这样索引可以保存所有变化的快照。

新增的文件总是要显示的添加到索引中来。对于那些之前已经提交过的文件，可以在commit命令中使用-a 选项达到提交到索引的目的。

## 1.2 Git安装

在Ubuntu上，你可以通过apt来安装git命令行工具

sudo apt-get install git-core

对于其他的Linux版本，请查看相关的软件包安装工具使用方法。msysgit项目提供了Windows版本的Git，地址是<http://code.google.com/p/msysgit/>

## 1.3 Git配置

你可以在.gitconfig文件中防止git的全局配置。文件位于用户的home目录。上述已经提到每次提交都会保存作者和提交者的信息，这些信息都可以保存在全局配置中。后续将会介绍配置用户信息、高亮显示和忽略特定的文件。

### 1.3.1 用户信息

通过如下命令来配置用户名和Email

git config --global user.name "Example Surname"

git config --global user.email "your.email@gmail.com"

# Set default so that all changes are always pushed to the repository

git config --global push.default "matching"

获取Git配置信息，执行以下命令：

git config --list

### 1.3.2 高亮显示

以下命令会为终端配置高亮

git config --global color.status auto

git config --global color.branch auto

### 1.3.3 忽略特定的文件

可以配置Git忽略特定的文件或者是文件夹。这些配置都放在.gitignore文件中。这个文件可以存在于不同的文件夹中，可以包含不同的文件匹配模式。为了让Git忽略bin文件夹，在主目录下放置.gitignore文件，其中内容为bin。

同时Git也提供了全局的配置，core.excludesfile。

### 1.3.4 使用.gitkeep来追踪空的文件夹

Git会忽略空的文件夹。如果你想版本控制包括空文件夹，根据惯例会在空文件夹下放置.gitkeep文件。其实对文件名没有特定的要求。一旦一个空文件夹下有文件后，这个文件夹就会在版本控制范围内。

## 1.4 开始操作Git

后续将通过一个典型的Git工作流来学习。在这个过程中，你会创建一些文件、创建一个本地的Git仓库、提交你的文件到这个仓库中。这之后，你会克隆一个仓库、在仓库之间通过pull和push操作来交换代码的修改。注释（以#开头）解释了命令的具体含义，让我们打开命令行开始操作吧。

### 1.4.1 创建内容

下面创建一些文件，它们会被放到版本控制之中

#Switch to home

cd ~/

# Create a directory

mkdir ~/repo01

# Switch into it

cd repo01

# Create a new directory

mkdir datafiles

# Create a few files

touch test01

touch test02

touch test03

touch datafiles/data.txt

# Put a little text into the first file

ls >test01

### 1.4.2 创建仓库、添加文件和提交更改

每个Git仓库都是放置在.git文件夹下.这个目录包含了仓库的所有历史记录，.git/config文件包含了仓库的本地配置。

以下将会创建一个Git仓库，添加文件倒仓库的索引中，提交更改。

# Initialize the local Git repository

git init

# Add all (files and directories) to the Git repository

git add .

# Make a commit of your file to the local repository

git commit -m "Initial commit"

# Show the log file

git log

### 1.4.3 diff命令与commit更改

通过git diff命令，用户可以查看更改。通过改变一个文件的内容，看看gitdiff命令输出什么，然后提交这个更改到仓库中

# Make some changes to the file

echo "This is a change" > test01

echo "and this is another change" > test02

# Check the changes via the diff command

git diff

# Commit the changes, -a will commit changes for modified files

# but will not add automatically new files

git commit -a -m "These are new changes"

### 1.4.4 Status, Diff 和 Commit Log

下面会向你展示仓库现有的状态以及过往的提交历史

# Make some changes in the file

echo "This is a new change" > test01

echo "and this is another new change" > test02

# See the current status of your repository

# (which files are changed / new / deleted)

git status

# Show the differences between the uncommitted files

# and the last commit in the current branch

git diff

# Add the changes to the index and commit

git add . && git commit -m "More chaanges - typo in the commit message"

# Show the history of commits in the current branch

git log

# This starts a nice graphical view of the changes

gitk --all

### 1.4.5 更正提交的信息 - git amend

通过git amend命令，我们可以修改最后提交的的信息。上述的提交信息中存在错误，下面会修改这个错误。

git commit --amend -m "More changes - now correct"

### 1.4.6 删除文件

如果你删除了一个在版本控制之下的文件，那么使用git add .不会在索引中删除这个文件。需要通过带-a选项的git commit命令和-A选项的git add命令来完成

# Create a file and put it under version control

touch nonsense.txt

git add . && git commit -m "a new file has been created"

# Remove the file

rm nonsense.txt

# Try standard way of committing -> will not work

git add . && git commit -m "a new file has been created"

# Now commit with the -a flag

git commit -a -m "File nonsense.txt is now removed"

# Alternatively you could add deleted files to the staging index via

git add -A .

git commit -m "File nonsense.txt is now removed"

## 1.5 远端仓库（remote repositories）

### 1.5.1 设置一个远端的Git仓库

我们将创建一个远端的Git仓库。这个仓库可以存储在本地或者是网络上。

远端Git仓库和标准的Git仓库有如下差别：一个标准的Git仓库包括了源代码和历史信息记录。我们可以直接在这个基础上修改代码，因为它已经包含了一个工作副本。但是远端仓库没有包括工作副本，只包括了历史信息。可以使用–bare选项来创建一个这样的仓库。

为了方便起见，示例中的仓库创建在本地文件系统上

# Switch to the first repository

cd ~/repo01

#

git clone --bare . ../remote-repository.git

# Check the content, it is identical to the .git directory in repo01

ls ~/remote-repository.git

### 1.5.2 推送更改到其他的仓库

做一些更改，然后将这些更改从你的第一个仓库推送到一个远端仓库

cd ~/repo01

echo "Hello, hello. Turn your radio on" > test01echo "Bye, bye. Turn your radio off" > test02

git commit -a -m "Some changes"

git push ../remote-repository.git

### 1.5.3 添加远端仓库

除了通过完整的URL来访问Git仓库外，还可以通过git remote add命令为仓库添加一个短名称。当你克隆了一个仓库以后，origin表示所克隆的原始仓库。即使我们从零开始，这个名称也存在。

# Add ../remote-repository.git with the name origin

git remote add origin ../remote-repository.git

# Again some changes

echo "I added a remote repo" > test02

# Commit

git commit -a -m "This is a test for the new remote origin"

# If you do not label a repository it will push to origin

git push origin

### 1.5.4 显示已有的远端仓库

通过以下命令查看已经存在的远端仓库

# Show the existing defined remote repositories

git remote

### 1.5.5 克隆仓库

通过以下命令在新的目录下创建一个新的仓库

# Switch to home

cd ~

# Make new directory

mkdir repo02

# Switch to new directory

cd ~/repo02

# Clone

git clone ../remote-repository.git .

### 1.5.6 拉取（Pull）更改

通过拉取，可以从其他的仓库中获取最新的更改。在第二个仓库中，做一些更改，然后将更改推送到远端的仓库中。然后第一个仓库拉取这些更改

# Switch to home

cd ~

# Switch to second directory

cd ~/repo02

# Make changes

echo "A change" > test01

# Commit

git commit -a -m "A change"

# Push changes to remote repository

# Origin is automatically maintained as we cloned from this repository

git push origin

# Switch to the first repository and pull in the changes

cd ~/repo01

git pull ../remote-repository.git/

# Check the changes

less test01

### 1.5.7 还原更改

如果在你的工作副本中，你创建了不想被提交的文件，你可以丢弃它。

# Create a new file with content

touch test04

echo "this is trash" > test04

# Make a dry-run to see what would happen

# -n is the same as --dry-run

git clean -n

# Now delete

git clean -f

你可以提取老版本的代码，通过提交的ID。git log命令可以查看提交ID

# Switch to home

cd ~/repo01

# Get the log

git log

# Copy one of the older commits and checkout the older revision via 译者注：checkout 后加commit id就是把commit的内容复制到index和工作副本中

git checkout commit\_name

如果你还未把更改加入到索引中，你也可以直接还原所有的更改

#Some nonsense change

echo "nonsense change" > test01

# Not added to the staging index. Therefore we can

# just checkout the old version

#译者注：checkout后如果没有commit id号，就是从index中拷贝数据到工作副本，不涉及commit部分的改变

git checkout test01

# Check the result

cat test01

# Another nonsense change

echo "another nonsense change" > test01

# We add the file to the staging index

git add test01

# Restore the file in the staging index

#译者注：复制HEAD所指commit的test01文件到index中

git reset HEAD test01

# Get the old version from the staging index

#译者注：复制index中test01到工作副本中

git checkout test01

#译者注，以上两条命令可以合并为git checkout HEAD test01

也可以通过revert命令进行还原操作

# Revert a commit

git revert commit\_name

即使你删除了一个未添加到索引和提交的文件，你也可以还原出这个文件

# Delete a file

rm test01

# Revert the deletion

git checkout test01

如果你已经添加一个文件到索引中，但是未提交。可以通过git resetfile 命令将这个文件从索引中删除

// Create a file

touch incorrect.txt

// Accidently add it to the index

git add .

// Remove it from the index

git reset incorrect.txt

// Delete the file

rm incorrect.txt

如果你删除了文件夹且尚未提交，可以通过以下命令来恢复这个文件夹 。译者注：即使已经提交，也可以还原

git checkout HEAD -- your\_dir\_to\_restore

译者注：checkout和reset这两个命令的含义是不同的，可以参阅这篇文章<http://marklodato.github.com/visual-git-guide/index-en.html>

### 1.5.8 标记

Git可以使用对历史记录中的任一版本进行标记。这样在后续的版本中就能轻松的找到。一般来说，被用来标记某个发行的版本。可以通过git tag命令列出所有的标记，通过如下命令来创建一个标记和恢复到一个标记

git tag version1.6 -m 'version 1.6'

git checkout <tag\_name>

## 1.6 分支、合并

### 1.6.1 分支

通过分支，可以创造独立的代码副本。默认的分支叫master。Git消耗很少的资源就能创建分支。Git鼓励开发人员多使用分支

下面的命令列出了所有的本地分支，当前所在的分支前带有\*号

git branch

如果你还想看到远端仓库的分支，可以使用下面的命令

git branch -a

可以通过下面的命令来创建一个新的分支

# Syntax: git branch <name> <hash>

# <hash> in the above is optional

# if not specified the last commit will be used

# If specified the corresponding commit will be used

git branch testing

# Switch to your new branch

git checkout testing

# Some changes

echo "Cool new feature in this branch" > test01

git commit -a -m "new feature"

# Switch to the master branch

git checkout master

# Check that the content of test01 is the old one

cat test01

### 1.6.2 合并

通过Merge我们可以合并两个不同分支的结果。Merge通过所谓的三路合并来完成。分别来自两个分支的最新commit和两个分支的最新公共commit.可以通过如下的命令进行合并

# Syntax: git merge <branch-name>

git merge testing

一旦合并发生了冲突，Git会标志出来，开发人员需要手工的去解决这些冲突。解决冲突以后，就可以将文件添加到索引中，然后提交更改

### 1.6.3 删除分支

删除分支的命令如下：

#Delete branch testing

git branch -d testing

# Check if branch has been deleted

git branch

### 1.6.4 推送（push）一个分支到远端仓库

默认的，Git只会推送匹配的分支的远端仓库。这意味在使用git push命令默认推送你的分支之前，需要手工的推送一次这个分支。

# Push testing branch to remote repository

git push origin testing

# Switch to the testing branch

git checkout testing

# Some changes

echo "News for you" > test01

git commit -a -m "new feature in branch"

# Push all including branch

git push

通过这种方式，你可以确定哪些分支对于其他仓库是可见的，而哪些只是本地的分支

## 1.7 解决合并冲突

如果两个不同的开发人员对同一个文件进行了修改，那么合并冲突就会发生。而Git没有智能到自动解决合并两个修改。

在这一节中，我们会首先制造一个合并冲突，然后解决它，并应用到Git仓库中。

下面会产生一个合并冲突

# Switch to the first directory

cd ~/repo01

# Make changes

touch mergeconflict.txt

echo "Change in the first repository" > mergeconflict.txt

# Stage and commit

git add . && git commit -a -m "Will create merge conflict 1"

# Switch to the second directory

cd ~/repo02

# Make changes

touch mergeconflict.txt

echo "Change in the second repository" > mergeconflict.txt

# Stage and commit

git add . && git commit -a -m "Will create merge conflict 2"

# Push to the master repository

git push

# Now try to push from the first directory

# Switch to the first directory

cd ~/repo01

# Try to push --> you will get an error message

git push

# Get the changes

git pull origin master

Git将冲突放在收到影响的文件中，文件内容如下：

<<<<<<< HEAD

Change in the first repository

=======

Change in the second repository

>>>>>>> b29196692f5ebfd10d8a9ca1911c8b08127c85f8

上面部分是你的本地仓库，下面部分是远端仓库。现在编辑这个文件，然后commit更改。另外的，你可以使用git mergetool命令

# Either edit the file manually or use

git mergetool

# You will be prompted to select which merge tool you want to use

# For example on Ubuntu you can use the tool "meld"

# After merging the changes manually, commit them

git commit -m "merged changes"

## 1.8 变基（Rebase）

### 1.8.1 在同一分支中应用Rebase Commit

通过rebase命令可以合并多个commit为一个。这样用户push更改到远端仓库的时候就可以先修改commit历史

接下来我们将创建多个commit，然后再将它们rebase成一个commit

# Create a new file

touch rebase.txt

# Add it to git

git add . && git commit -m "rebase.txt added to index"

# Do some silly changes and commit

echo "content" >> rebase.txt

git add . && git commit -m "added content"

echo " more content" >> rebase.txt

git add . && git commit -m "added more content"

echo " more content" >> rebase.txt

git add . && git commit -m "added more content"

echo " more content" >> rebase.txt

git add . && git commit -m "added more content"

echo " more content" >> rebase.txt

git add . && git commit -m "added more content"

echo " more content" >> rebase.txt

git add . && git commit -m "added more content"

# Check the git log message

git log

我们合并最后的七个commit。你可以通过如下的命令交互的完成

git rebase -i HEAD~7

这个命令会打开编辑器让你修改commit的信息或者 squash/ fixup最后一个信息.Squash会合并commit信息而fixup会忽略commit信息（待理解）

### 1.8.2 Rebasing多个分支

你也可以对两个分支进行rebase操作。如下所述，merge命令合并两个分支的更改。rebase命令为一个分支的更改生成一个补丁，然后应用这个补丁到另一分支中

使用merge和rebase，最后的源代码是一样的，但是使用rebase产生的commit历史更加的少，而且历史记录看上去更加的线性

# Create new branch

git branch testing

# Checkout the branch

git checkout testing

# Make some changes

echo "This will be rebased to master" > test01

# Commit into testing branch

git commit -a -m "New feature in branch"

# Rebase the master

git rebase master

### 1.8.3 Rebase最佳实践

在push更改到其他的Git仓库之前，我们需要仔细检查本地分支的commit历史

在Git中，你可以使用本地的commit。开发人员可以利用这个功能方便的回滚本地的开发历史。但是在push之前，需要观察你的本地分支历史，是否其中有些commit历史对其他用户来说是无关的

如果所有的commit历史都跟同一个功能有关，很多情况下，你需要rebase这些commit历史为一个commit历史。

交互性的rebase主要就是做重写commit历史的任务。这样做是安全的，因为commit还没有被push到其它的仓库。这意味着commit历史只有在被push之前被修改。

如果你修改然后push了一个已经在目标仓库中存在的commit历史，这看起来就像是你实现了一些别人已经实现的功能

### 1.8.4 创建和应用补丁

一个补丁指的是一个包含对源代码进行修改的文本文件。你可以将这个文件发送给某人，然后他就可以应用这个补丁到他的本地仓库。

下面会创建一个分支，对这个分支所一些修改，然后创建一个补丁，并应用这个补丁到master分支

# Create a new branch

git branch mybranch

# Use this new branch

git checkout mybranch

# Make some changes

touch test05

# Change some content in an existing file

echo "New content for test01" >test01

# Commit this to the branch

git add .

git commit -a -m "First commit in the branch"

# Create a patch --> git format-patch master

git format-patch origin/master

# This created patch 0001-First-commit-in-the-branch.patch

# Switch to the master

git checkout master

# Apply the patch

git apply 0001-First-commit-in-the-branch.patch

# Do your normal commit in the master

git add .

git commit -a -m "Applied patch"

# Delete the patch

rm 0001-First-commit-in-the-branch.patch

## 1.9 定义同名命令

Git允许你设定你自己的Git命令。你可以给你自己常用的命令起一个缩写命令，或者合并几条命令道一个命令上来。

下面的例子中，定义了git add-commit 命令，这个命令合并了git add . -A 和git commit -m 命令。定义这个命令后，就可以使用git add-commit -m"message" 了.

git config --global alias.add-commit '!git add . -A && git commit'

但是非常不幸，截止写这篇文章之前，定义同名命令在msysGit中还没有支持。同名命令不能以！开始。

## 1.10 放弃跟踪文件

有时候，你不希望某些文件或者文件夹被包含在Git仓库中。但是如果你把它们加到.gitignore文件中以后，Git会停止跟踪这个文件。但是它不会将这个文件从仓库中删除。这导致了文件或者文件夹的最后一个版本还是存在于仓库中。为了取消跟踪这些文件或者文件夹，你可以使用如下的命令

# Remove directory .metadata from git repo

git rm -r --cached .metadata

# Remove file test.txt from repo

git rm --cached test.txt

这样做不会将这些文件从commit历史中去掉。如果你想将这些文件从commit历史中去掉，可以参考git filter-branch命令

## 1.11 其他有用的命令

下面列出了在日常工作中非常有用的Git命令

有用的Git命令

| **命令** | **描述** |
| --- | --- |
| git blame filename | 谁创建了或者是修改了这个文件 |
| git checkout -b mybranch | 以上上个commit信息为起点，创建一条 |
| master~1 | 新的分支 |

## 1.12 安装Git服务

如上所述，我们的操作不需要Git服务。我可以只使用文件系统或者是Git仓库的提供者，像Github或Bitbucket。但是，有时候，拥有一个自己的服务是比较方便的，在ubuntu下安装一个服务相对来说是比较容易的

确定你已经安装了ssh

apt-get install ssh

如果你还没有安装Git服务，安装它

sudo apt-get install git-core

添加一个名为git的用户

sudo adduser git

然后使用git用户进行登陆，创建一个空的仓库

# Login to server

# to test use localhost

ssh git@IP\_ADDRESS\_OF\_SERVER

# Create repository

git init --bare example.git

现在你就可以向远端的仓库提交变更了

mkdir gitexample

cd gitexample

git init

touch README

git add README

git commit -m 'first commit'

git remote add origin git@IP\_ADDRESS\_OF\_SERVER:example.git

git push origin master

## 1.13 在线的远端仓库

### 1.13.1 克隆远端仓库

Git支持远端的操作。Git支持多种的传输类型，Git自带的协议就叫做git。下面的的命令通过git协议从克隆一个仓库

git clone git@github.com:vogella/gitbook.git

同样的，你可以通过http协议来克隆仓库

# The following will clone via HTTP

git clone http://vogella@github.com/vogella/gitbook.git

### 1.13.2 添加远端仓库

如果你克隆了一个远端仓库，那么原先的仓库就叫做origin

你可以push修改到origin中，通过 git push origin 命令. 当然，push到一个远端的仓库需要对仓库的写权限

你可以通过git remote add name gitrepo 命令添加多个仓库。例如，你可以通过http协议再次添加之前clone过来的仓库:

// Add the https protocol

git remote add githttp https://vogella@github.com/vogella/gitbook.git

### 1.13.3 通过http和代理服务器进行远端操作

如果你的防火墙屏蔽了出http以外的所有协议，那么使用http协议来获取仓库是非常好的方法。.

Git同样支持通过代理服务器使用http协议。下面的Git命令会展示这一点。你可以为所有的程序设置代理服务器或者只是为Git服务提供。

下面的例子用到了环境变量

# Linux

export http\_proxy=http://proxy:8080

# On Windows

# Set http\_proxy=http://proxy:8080

git clone http://dev.eclipse.org/git/org.eclipse.jface/org.eclipse.jface.snippets.git

# Push back to the origin using http

git push origin

下面的例子只是用到了Git的配置

// Set proxy for git globally

git config --global http.proxy http://proxy:8080

// To check the proxy settings

git config --get http.proxy

// Just in case you need to you can also revoke the proxy settings

git config --global --unset http.proxy

## 1.14 Git服务提供商

除了架设自己的服务，你也可以使用Git服务提供商提供的服务。最流行的Git服务提供网站是GitHub和Bitbucket。它们都提供了有限制的免费服务

### 1.14.1 GitHub

可以通过 <https://github.com/> 访问GitHub. GitHub上所有的公开仓库都是免费的。如果你想在上面使用私有的仓库，那么就需要付费给GitHub

GitHub需要你创建ssh的公钥私钥。生成一份Ubuntu的公钥私钥可以访问 sshkey creation in Ubuntu ，Windows环境可以访问msysgit ssh key generation.

在GitHub上创建一个账户和一个仓库以后。你会收到如何将你的项目上传到GitHUb的指南，其中的命令大致如下：

Global setup:

Set up git

git config --global user.name "Your Name"

git config --global user.email your.email@gmail.com

Next steps:

mkdir gitbook

cd gitbook

git init

touch README

git add README

git commit -m 'first commit'

git remote add origin git@github.com:vogella/gitbook.git

git push -u origin master

Existing Git Repo?

cd existing\_git\_repo

git remote add origin git@github.com:vogella/gitbook.git

git push -u origin master

### 1.14.2 Bitbucket

可以通过 <https://bitbucket.org/> 访问Bitbucket. Bitbucket 提供了无限制了公共仓库和只能有五个人访问的私有仓库。如果你需要超过五个人访问私有仓库，就需要付费给Bitbucket

## 1.15 Git的图形接口

这个教程主要说明Git命令行的使用。完成了这个教程以后，你可能想要找到一个Git的图形工具

Git提供了两个图形工具。 gitk能够展示仓库的历史信息、git gui 让你可以通过编辑器来完成Git操作

Eclipse EGit 项目提供了Git与Eclipse的集成，在最新的Eclipse版本中可以找到

## 1.16 Kindle版本教程

这个教程提供了Kindle版本

Kindle Edition

## 1.17 问题与讨论

在提出问题之前，请先查看 vogella FAQ. 如果你有任何的问题或者是从文章中找到错误，那么可以使用www.vogella.com Google Group. 我自己写了一个简短的列表 how to create good questions 可能会对你有用.

## 1.18 链接和文章

1.[Git相关学习资源](http://download.csdn.net/detail/tangbin330/5625037)

Git homepage

EGit - Teamprovider for Eclipse

Video with Linus Torwalds on Git

Progit book - Free Git book

Video casts about Git

<http://code.google.com/p/msysgit/> Git on Windows

<http://github.com/guides/git-cheat-sheet> Git Cheat Sheets

Author: hic[<lishuo.os.ds@gmail.com>](mailto:lishuo.os.ds@gmail.com)

Date: 2012-08-26 日

HTML generated by org-mode 6.33x in emacs 23