长城总结：

**Windows 上安装**

在 Windows 上安装 Git 同样轻松，有个叫做 msysGit 的项目提供了安装包，可以从 Google Code 的页面上下载安装文件（.exe）：

http://code.google.com/p/msysgit

mkdir [file] 新建文件夹

touch a.txt b.txt //创建多个文件 /另外echo也可以他们都是linux的命令。例如echo的例子：$ echo "this is hello " >> hello 这样就建立了一个hello文件夹并填充了内容。

第一个要配置的是你个人的用户名称和电子邮件地址。这两条配置很重要，每次 Git 提交时都会引用这两条信息，说明是谁提交了更新，所以会随更新内容一起被永久纳入历史记录：

$ git config --global user.name "John Doe"

$ git config --global user.email johndoe@example.com

如果用了 --global 选项，那么更改的配置文件就是位于你用户主目录下的那个，以后你所有的项目都会默认使用这里配置的用户信息。如果要在某个特定的项目中使用其他名字或者电邮，只要去掉 --global 选项重新配置即可，新的设定保存在当前项目的 .git/config 文件里。

【第一】：从当前目录初始化

要对现有的某个项目开始用 Git 管理，只需到此项目所在的目录，执行：

$ git init

如果当前目录下有几个文件想要纳入版本控制，需要先用 git add 命令告诉 Git 开始对这些文件进行跟踪，然后提交：

$ git add \*.c

$ git add README

$ git commit -m 'initial project version'

【第二】：克隆仓库的命令格式为 git clone [url]。比如，要克隆 Ruby 语言的 Git 代码仓库 Grit，可以用下面的命令：

$ 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，并从同步后的仓库中拉出所有的数据，取出最新版本的文件拷贝。如果进入这个新建的 grit 目录，你会看到项目中的所有文件已经在里边了，准备好后续的开发和使用。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

（再次说明下，克隆操作会自动使用默认的 master 和 origin 名字。

【第三】：要确定哪些文件当前处于什么状态，可以用 git status 命令。如果在克隆仓库之后立即执行此命令，会看到类似这样的输出：

$ git status

# On branch master

nothing to commit (working directory clean)

【第四】：查看已暂存和未暂存的更新

要查看尚未暂存的文件更新了哪些部分，不加参数直接输入 git diff：

若要看已经暂存起来的文件和上次提交时的快照之间的差异，可以用 git diff --cached 命令。（Git 1.6.1 及更高版本还允许使用 git diff --staged，效果是相同的，但更好记些。）

### 【第五】：跳过使用暂存区域

尽管使用暂存区域的方式可以精心准备要提交的细节，但有时候这么做略显繁琐。Git 提供了一个跳过使用暂存区域的方式，只要在提交的时候，给 git commit 加上 -a 选项，Git 就会自动把所有已经跟踪过的文件暂存起来一并提交，从而跳过 git add 步骤：

$ git status

# On branch master

#

# Changed but not updated:

#

# modified: benchmarks.rb

#

$ git commit -a -m 'added new benchmarks'

[master 83e38c7] added new benchmarks

1 files changed, 5 insertions(+), 0 deletions(-)

【第六】：移除文件

可以用 git rm 命令完成此项工作，并连带从工作目录中删除指定的文件，这样以后就不会出现在未跟踪文件清单中了。

如果只是简单地从工作目录中手工删除文件，运行 git status 时就会在 “Changed but not updated” 部分（也就是\_未暂存\_清单）看到：

$ rm grit.gemspec

$ git status

# On branch master

#

# Changed but not updated:

# (use "git add/rm <file>..." to update what will be committed)

#

# deleted: grit.gemspec

#

然后再运行 git rm 记录此次移除文件的操作：

$ git rm grit.gemspec

rm 'grit.gemspec'

$ git status

# On branch master

#

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#

# deleted: grit.gemspec

#

要移除跟踪但不删除文件用$ git rm --cached readme.txt

，以便稍后在 .gitignore 文件中补上，用 --cached 选项即可：

另外$ git rm \\*~

表示会递归删除当前目录及其子目录中所有 ~ 结尾的文件

【第七】：重命名文件

关于重命名操作的说明：

$ git mv README.txt README

$ git status

# On branch master

# Your branch is ahead of 'origin/master' by 1 commit.

#

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#

# renamed: README.txt -> README

#

其实，运行 git mv 就相当于运行了下面三条命令：

$ mv README.txt README

$ git rm README.txt

$ git add README

【第八】查看提交历史

git log

git log 会按提交时间列出所有的更新，最近的更新排在最上面。每次更新都有一个 SHA-1 校验和、作者的名字和电子邮件地址、提交时间，最后缩进一个段落显示提交说明。

git log 有许多选项可以帮助你搜寻感兴趣的提交，接下来我们介绍些最常用的。

我们常用 -p 选项展开显示每次提交的内容差异，用 -2 则仅显示最近的两次更新：

--stat，仅显示简要的增改行数统计：

--pretty 选项，可以指定使用完全不同于默认格式的方式展示提交历史。比如用 oneline 将每个提交放在一行显示，这在提交数很大时非常有用。另外还有 short，full 和 fuller 可以用，展示的信息或多或少有些不同。

另外 format，可以定制要显示的记录格式，这样的输出便于后期编程提取分析，像这样：

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

ca82a6d - Scott Chacon, 11 months ago : changed the verison number

085bb3b - Scott Chacon, 11 months ago : removed unnecessary test code

a11bef0 - Scott Chacon, 11 months ago : first commit

表 2-1 列出了常用的格式占位符写法及其代表的意义。

选项 说明

%H 提交对象（commit）的完整哈希字串

%h 提交对象的简短哈希字串

%T 树对象（tree）的完整哈希字串

%t 树对象的简短哈希字串

%P 父对象（parent）的完整哈希字串

%p 父对象的简短哈希字串

%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

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。

以上只是简单介绍了一些 git log 命令支持的选项。还列出了一些其他常用的选项及其释义。

选项 说明

-p 按补丁格式显示每个更新之间的差异。

--stat 显示每次更新的文件修改统计信息。

--shortstat 只显示 --stat 中最后的行数修改添加移除统计。

--name-only 仅在提交信息后显示已修改的文件清单。

--name-status 显示新增、修改、删除的文件清单。

--abbrev-commit 仅显示 SHA-1 的前几个字符，而非所有的 40 个字符。

--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）。

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：

（选项 说明

-(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。

）

$ git log --since=2.weeks

[详解:

(第一：)git log -p'它不但会打印出提交的时间、版本号、人员等，还会将具体的代码修改部分打印出来;'-n'其中n表示一个数字，这表示打印出具体的几个日志，例如 'git -p -1'正如我的Git配置文件中设置的onelog别名的内容一样，就表示打印出当前最新的一次日志记录及具体修改内容;'--since="时间/日期"'，'--until="时间/日期"'表示你希望查找某个日期段的日志记录，例如'git log --since="2 days ago" --until="1 hour ago"'就表示你希望查找两天前到一小时前的日志记录，Git是足够聪明的，它可以将类似于'2 days ago'和'1 hour ago'这种表示时间的英语转化为具体的时间数字;有的时候，你不希望翻很多页才可以看到所有的日志，你只希望看到简短的说明，那么Git为你提供打印格式的定制'git --pretty=格式种类'，其中格式种类有full、short、oneline等，例如'git log pretty=oneline'就会将每条代码历史记录放在一行里，看起来简单明了.

(第二：) 　　git stash：在第一篇中，我举了一个使用branch解决紧急任务切换的问题，其实stash命令也可以很好的解决这样的问题。当你不想提交当前完成了一半的代码，但是却不得不修改一个紧急Bug，那么使用'git stash'就可以将你当前未提交到本地(和服务器)的代码推入到Git的栈中，这时候你的工作区间和上一次提交的内容是完全一样的，所以你可以放心的修 Bug，等到修完Bug，提交到服务器上后，再使用'git stash apply'将以前一半的工作应用回来。也许有的人会说，那我可不可以多次将未提交的代码压入到栈中?答案是可以的。当你多次使用'git stash'命令后，你的栈里将充满了未提交的代码，这时候你会对将哪个版本应用回来有些困惑，'git stash list'命令可以将当前的Git栈信息打印出来，你只需要将找到对应的版本号，例如使用'git stash apply stash@{1}'就可以将你指定版本号为stash@{1}的工作取出来，当你将所有的栈都应用回来的时候，可以使用'git stash clear'来将栈清空

git stash: 备份当前的工作区的内容，从最近的一次提交中读取相关内容，让工作区保证和上次提交的内容一致。同时，将当前的工作区内容保存到Git栈中。  
git stash pop: 从Git栈中读取最近一次保存的内容，恢复工作区的相关内容。由于可能存在多个Stash的内容，所以用栈来管理，pop会从最近的一个stash中读取内容并恢复。  
git stash list: 显示Git栈内的所有备份，可以利用这个列表来决定从那个地方恢复。  
git stash clear: 清空Git栈。此时使用gitg等图形化工具会发现，原来stash的哪些节点都消失了

]

【第九】使用图形化工具查阅提交历史

项目工作目录中输入 gitk 命令后，就会启动。随 Git 一同发布的 gitk 就是这样一种工具。它是用 Tcl/Tk 写成的，基本上相当于 git log 命令的可视化版本，凡是 git log 可以用的选项也都能用在 gitk 上。

【第十】撤消操作

### 修改最后一次提交

有时候我们提交完了才发现漏掉了几个文件没有加，或者提交信息写错了。想要撤消刚才的提交操作，可以使用 --amend 选项重新提交：

$ git commit --amend

如果刚才提交时忘了暂存某些修改，可以先补上暂存操作，然后再运行 --amend 提交：

$ git commit -m 'initial commit'

$ git add forgotten\_file

$ git commit --amend

【第十一】取消已经暂存的文件

括号中有提示，可以使用 git reset HEAD <file>... 的方式取消暂存。

git-reset命令有三个选项：

--mixed、

--soft和

--hard。

我们在日常使用中仅使用前两个选项；第三个选项由于杀伤力太大，容易损坏项目仓库，需谨慎使

用。

--mixed是 git-reset的默认选项，它的作用是重置索引内容，将其定位到指定

的项目版本，而不改变你的工作树中的所有内容，只是提示你有哪些文件还未更新。不带任何参数的git reset，即时这种方式，它回退到某个版本，只保留源码，回退commit和index信息

--soft选项既不触动索引的位置，也不改变工作树中的任何内容，但是会要求

它们处于一个良好的次序之内。该选项会保留你在工作树中的所有更新并使之处于

待提交状态。回退到某个版本，只回退了commit的信息，不会恢复到index file一级。如果还要提交，直接commit即可

git reset –hard：彻底回退到某个版本，本地的源码也会变为上一个版本的内容。

如果欲查看git-status命令。

git-reset命令对工作树的影响，可使用

#回退所有内容到上一个版本

git reset HEAD^

#回退a.py这个文件的版本到上一个版本

git reset HEAD^ a.py

#向前回退到第3个版本

git reset –soft HEAD~3

#将本地的状态回退到和远程的一样

git reset –hard origin/master

回退到某个版本

git reset 057d

#回退到上一次提交的状态，按照某一次的commit完全反向的进行一次commit

git revert HEAD

｛长城自己总结：当本地回退到某个版本后，通过 git push –f origin master 可以强制更新远程仓库｝

【第十二】取消对文件的修改

取消修改，回到之前的状态（也就是修改之前的版本）

"git checkout -- <file>..."

【第十三】远程仓库的使用

查看当前的远程库，可以用 git remote 命令。它会列出每个远程库的简短名字。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。

也可以加上 -v 选项（译注：此为 –verbose 的简写，取首字母），显示对应的克隆地址：

$ git remote -v

origin git://github.com/schacon/ticgit.git

添加远程仓库git remote add [shortname] [url]

$ git remote

origin

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

$ git remote -v

origin git://github.com/schacon/ticgit.git

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

从远程仓库抓取数据$ git fetch [remote-name]

相当于是从远程获取最新版本到本地，不会自动merge  
 git fetch origin master  
git log -p master..origin/master  
git merge origin/master    以上命令的含义：  
   首先从远程的origin的master主分支下载最新的版本到origin/master分支上  
   然后比较本地的master分支和origin/master分支的差别  
   最后进行合并.

# 合并b   
git rebase b   
# 处理完冲突继续合并   
git rebase --continue   
# 跳过   
git rebase --skip   
# 取消合并   
git rebase –abort

[网上视频说的是：git fetch origin 然后git rebase origin/master, 然后 git rebase - –continue,然后git push。如果有冲突情况则步骤是：git fetch origin ,git rebase origin/master, 删除冲突内容后 git add . ,最后git push即可。 长城测试后提醒：如果git push 后提示error:failed to push some refs to … …的错误 只需要这样 git push origin –f 这里的-f 就是--force]

另外 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.

推送数据到远程仓库git push [remote-name] [branch-name]。$ git push origin master.

查看远程仓库信息git remote show [remote-name]

远程仓库的重命名git remote rename比如想把 pb 改成 paul，可以这么运行：

$ git remote rename pb paul

$ git remote

origin

paul

远程仓库的删除运行 git remote rm 命令如删除paul：

$ git remote rm paul

$ git remote

Origin

【第十四】分支

查看本地分支：$ **git** branch

查看远程分支：$ **git** branch -r

创建本地分支：$ **git** branch [name] ----注意新分支创建后不会自动切换为当前分支

切换分支：$ **git** checkout [name]

创建新分支并立即切换到新分支：$ **git** checkout -b [name]

删除分支：$ **git** branch -d [name] ---- -d选项只能删除已经参与了合并的分支，对于未有合并的分支是无法删除的。如果想强制删除一个分支，可以使用-D选项

合并分支：$ **git** merge [name] ----将名称为[name]的分支与当前分支合并

创建远程分支(本地分支push到远程)：$ git push origin [name]

删除远程分支：$ git push origin :heads/[name]

我从master分支创建了一个issue5560分支，做了一些修改后，使用**git push** origin master提交，但是显示的结果却是'Everything up-to-date'，发生问题的原因是**git push** origin master 在没有track远程分支的本地分支中默认提交的master分支，因为master分支默认指向了origin master 分支，这里要使用**git push** origin issue5560：master 就可以把issue5560推送到远程的master分支了

新建一个 testing 分支，可以使用 git branch 命令：

$ git branch testing

git checkout 命令。我们现在转换到新建的 testing 分支：

$ git checkout testing

新建的分支取名为 iss53。要新建并切换到该分支，运行 git checkout 并加上 -b 参数：$ git checkout -b iss53

Switched to a new branch "iss53"相当于下面这两条命令：

$ git branch iss53

$ git checkout iss53

用 git merge 命令来进行合并。

因为现在 hotfix 分支和 master 指向相同的提交，现在没什么用了，可以先删掉它。使用 git branch 的 -d 选项表示删除：

$ git branch -d hotfix

Deleted branch hotfix (3a0874c).

**冲突的合并**

解决冲突的办法无非是二者选其一或者由你亲自整合到一起。运行 git add 将把它们标记为已解决（resolved）。因为一旦暂存，就表示冲突已经解决。

**分支管理**

git branch 命令它会给出当前所有分支的清单：

$ git branch

iss53

\* master

testing

若要查看各个分支最后一次 commit 信息，运行 git branch –v。

要从该清单中筛选出你已经（或尚未）与当前分支合并的分支，可以用 --merge 和 --no-merged 选项.

git push [远程名] [本地分支]:[远程分支] 语法，如果省略 [本地分支] ，那就等于是在说“在这里提取空白然后把它变成[远程分支] ”。(例如：$ git push origin test:test           // 提交本地test分支作为远程的test分支。如果想删除远程的分支呢？$ git push origin :test         // 刚提交到远程的test将被删除，但是本地还会保存的，不用担心)

【十五：】一个可以保存和恢复工作进度的命令

在切换到新的工作分支之前，执行

git stash保存工作进度，工作区就会变得非常干净，

然后就可以切换到新的分支中了。

$ git stash

$ git checkout <new\_branch>

新的工作分支修改完毕后，再切换回当前分支，调用

git stash pop命令则可恢复之前保

存的工作进度。

$ git checkout <orignal\_branch>

$ git stash pop

【十六：】忽略一些文件、文件夹不提交。

在仓库根目录下创建名称为“.gitignore”的文件，写入不需要的文件夹名或者文件，每个元素占一行即可，如

target

bin

\*.db

【小窍门】

（1）命令的选项也可以用这种方式自动完成，其实这种情况更实用些。比如运行 git log 的时候忘了相关选项的名字，可以输入开头的几个字母，然后敲 Tab 键看看有哪些匹配的。

（2） 另外对于命令不妨设置别名如可以用 git config 为命令设置别名。来看看下面的例子：

[user]

name = xcc

email = [changchengxue@gmail.com](mailto:changchengxue@gmail.com)

[alias]

co = checkout

ci = commit -a

st = status

br = branch

oneline = log --pretty=oneline --since='2 days ago'

onelog = log -p -1

[remote "myremote"]

url =git@github.com:changchengxu/testApp.git

fetch=+refs/heads/\*:refs/remotes/a/\*

[color]

status = auto

branch = auto

ui = auto

[receive]

denyCurrentBranch = ignore

现在，如果要输入 git commit 只需键入 git ci 即可

（3）我们希望运行某个外部命令，而非 Git 的附属工具，只需要在命令前加上 ! 就行。例如

我们可以设置用 git visual 启动 gitk：

$ git config --global alias.visual ‘!gitk‘

---------------------------------------------------------------------------------------------------------------------------------

**Goto directory**

$ cd / goto /

$ cd ~ goto “/home/user/”

$ cd .. goto up level directory

$ cd - goto previous directory

**Display current directory**

$ pwd

/home/user/workspace

**Copy**

$ cp source\_file dest\_file

**Remove**

$ rm dest\_file

$ rm dest\_dir -r

**Move**

$ mv source\_file dest\_dir/dest\_file

**Rename**

$ mv source\_file dest\_file

**Terminal**

vi, vim

$ vi text\_file\_sample

Goto edit mode: press 'i' or 'insert' key

Press 'ESC' button to quit 'edit mode'

Press ':' go to command mode

Commands:

* + - w (write) save file
    - q (quit) quit vi
    - wq save and quit
    - q! force quit without save
    - / find word

(4)ls 查看当前文件夹下文件目录。

ls命令是linux最常用的命令，用来显示文件或者目录的相关信息。一般会搭配以下参数常用：

-a：显示全部文件，与隐藏文件(开头为.的文件)一起列出来。

-l：列出长数据串，包括文件属性。

-t：按时间排序。

-h：列出文件大小。

(5)cd或git中如果打开的文件名有空格，可以用“”即可打开。

(6)查看当前目录(pwd)  
查看当前所在系统目录命令：pwd

例如：如果当前目录在/mas之下，执行pwd，那么就会显示当前路径/mas。

（7）直接查看文件内容命令：cat

（8）q：立即退出

**生成一个新的 SSH Key**邮箱为你之前注册时的邮箱$ssh-keygen -t rsa -C "email" email必须要填写

创建远程分支：

git remote add origin [git@github.com:changchengxu/sec-repo.git](mailto:git@github.com:changchengxu/sec-repo.git)

【远端上传步骤：

**Global setup:**

git config –global user.name “your name”

git config –global user.email [changchengxue@gmail.com](mailto:changchengxue@gmail.com)

**Next steps**

Mkdir sec-repo

Cd sec –repo

Git init

Touch readme

Git add readm

Git commit –m ‘first commit ‘

Git remote add soc git @github.com:changchengxu/soc.git

Git push –u soc master (对于 git add –u .参数 -u，表示将各种变动也添加进来,这里应该也是)

】

ssh://git@10.96.5.11/repositories/hdic3101/soc\_nds //这个是hdic的远端

要查阅的网站：

<http://www.cnblogs.com/wojilu/archive/2011/11/16/2250721.html>

以及

<http://blog.csdn.net/zzulp/article/details/6238527#Fetchingbranchesfromotherrepositories.2Bg7dT1lF2TtZySGcsXpN2hFIGZS8->

以及

<http://blog.csdn.net/wojilu/article/details/6976230>

**疑难问题解答**：

第一：error: failed to push some refs to … …

解答：git pull， 在自动merge或手动merge冲突， 再次git push 成功解决问题。

vi /home/git/gitconfig.rc 或

git config --global receive.denyDeletes true/false （设置为true则服务器分支不可以删除，false则为可以删除）长城没有测试，不知道可行吗