CVS及SVN都是集中式的版本控制系统，而Git是分布式版本控制系统

判断linux系统上是否安装了git
未命名图片.png git 
usage: git [ - -version I [ - -helpl C -C cpath>l [ -c name=valuel 
[ - [--html-path] [ - -man-path] [--tnfo-path] 
•paginate I 
• •no-pager) t • •no-replace-objects] t • •bare] 
«command> [«args»] 
(8!0.• it help tutorial) 
clone 
tntt 
everyday) 
HEAD 
bisect 
show 
status 
checkout 
commit 
diff 
merge 
rebase 
( e-Æ: gtt hel workflows) 
fetch 
pull 
push 
bug 
help 

创建版本库（目录）
这个目录里面的所有⽂文件都可以被Git管理起来，每个⽂文件的修改、删除，Git都能跟踪，以便任何时刻都可以追踪历史，或者在将来某个时刻可以“还原”。

①创建⼀一个版本库非常简单，首先，选择⼀一个合适的地⽅方，创建⼀一个空目录：
$ mkdir learngit
$ cd learngit
$ pwd
/Users/michael/learngit
pwd命令用于显示当前目录
②通过git init命令把这个目录变成Git可以管理的仓库：
$ git init
Initialized empty Git repository in /Users/michael/learngit/.git/

UTF-8编码

把一个文件放到Git仓库只需要两步：
主要用到的命令：
git add.

git  commit -m "……"

git push

git pull

git clone  只进行一次，即从远程库中得到当前工程
所以不用执行 git remote add origin git@github.com:xinwenmei/Test.git
  
把一个文件放到Git仓库只需要两步：
第一步，⽤用命令git add告诉Git，把文件添加到仓库：
$ git add readme.txt
第二步，⽤用命令git commit告诉Git，把文件提交到仓库：
$ git commit -m "wrote a readme file"
[master (root-commit) cb926e7] wrote a readme file
1 file changed, 2 insertions(+)
create mode 100644 readme.txt
git commit命令，-m后⾯面输入的是本次提交的说明

小结
现在总结一下今天学的两点内容：
初始化一个Git仓库，使用git init命令。
添加文件到Git仓库，分两步：
• 第一步，使⽤用命令git add ，注意，可反复多次使用，添加多个文件；
• 第二步，使⽤用命令git commit，完成。


未命名图片.png plurn@0ESKTop-TMOG97M MINSW64 -/learngit (master) 
S git status 
On branch master 
Changes not staged for commit: 
(use iigit add 
to update what will be committed) 
(use "git checkout 
to discard changes in working di rectory) 
modifi ed: 
readme txt 
o changes added to commit (use "git add" and/or "git commit -a") 
plun0DÉSKTOP-TMDG97M MINGW64 -/learngit (master) 
S git diff readme.txt 
diff --git a/readme.txt b/readme.txt 
index dS036c1. .013b5bc 100644 
a/readme. txt 
+++ b/readme. txt 
-1,2 2 
*Git is a distributed version control system. 
Cit is free software. 
\ No newline at end of file 

git status：查看状态（发现readme.txt文件被修改了）
git diff：查看difference

小结
• 要随时掌握工作区的状态，使用git status命令。
• 如果git status告诉你有文件被修改过，用git diff可以查看修改内容。

Git中，每当文件修改到一定程度的时候，就可以“保存一个快照”，这个快照在Git中被称为commit。一旦你把文件改乱了，或者误 删了文件，还可以从最近的一个commit恢复，然后继续工作。

未命名图片.png plumßDESKTOP-TMDG97M MINGW64 -/learngit (master) 
S git log 
commit 2d04d82b7dd54bd911c050427e9d886a73db8387 (HEAD 
Author: x•inwenmei <12062189640qq . com> 
Date: Tue Apr 3 2018 +0800 
append CPL 
commit 4ef8ec2e632213d14dcd7855af1eee9609266f63 
Author: xinwenmei . com> 
Date: Tue Apr 3 09:07 : 40 2018 +0800 
add distributed 
commit e9f5a174f441cf9cce8ea991c2d4b2bf77725b0a 
Author: x•inwenmei . com> 
Date: Tue Apr 3 08 : 54: 47 2018 +0800 
wrote a readme file 
-> master) 
  
未命名图片.png plumßDESKTOP-TMDG97M MINGW64 -/learngit (master) 
S git log 
commit 2d04d82b7dd54bd911c050427e9d886a73db8387 (HEAD 
Author: x•inwenmei <12062189640qq . com> 
Date: Tue Apr 3 2018 +0800 
append CPL 
commit 4ef8ec2e632213d14dcd7855af1eee9609266f63 
Author: xinwenmei . com> 
Date: Tue Apr 3 09:07 : 40 2018 +0800 
add distributed 
commit e9f5a174f441cf9cce8ea991c2d4b2bf77725b0a 
Author: x•inwenmei . com> 
Date: Tue Apr 3 08 : 54: 47 2018 +0800 
wrote a readme file 
-> master) 
在Git中，用个git log 命令查看历史记录（提交日志）

未命名图片.png pl um@0ESKTOP-TMOG97M MING"'64 -/learngit (master) 
S git reset --hard HEADA 
HEAD is now at 4efSec2 add distributed 
plum@DESKTOP-TMDG97M MINGW64 -/learngit (master) 
S cat readme. txt 
Git is a distributed version control system. 
,Git is free software. 


未命名图片.png S git ref log 
2d04dS2 (HEAD 
4ef8ec2 
2d04dS2 (HEAD 
4ef8ec2 
e9f5a17 
master) 
reset: 
moving to HEADA 
• reset: 
master) 
convnit: 
commit: add distributed 
(initial): wrote 
commit 
moving to 2d04d82 
append CPL 
a readme file 
Git提供了一个命令git reflog用来记录你的每一次命令
第三行显⽰示“append GPL”的commit id是2d04d82

现在总结一下：
• HEAD指向的版本就是当前版本，因此，Git允许我们在版本的历史之间穿梭，使用命令git reset --hard commit_id。
• 穿梭前，用git log可以查看提交历史，以便确定要回退到哪个版本。
• 要重返未来，用git reflog查看命令历史，以便确定要回到未来的哪个版本。

Git和其他版本控制系统如SVN的一个不同之处就是有暂存区的概念。
工作区（Working Directory）：就是你在电脑里能看到的目录，比如我的learngit文件夹
就是一个工作区：
版本库（Repository）：工作区有一个隐藏目录“.git”，这个不算工作区，而是Git的版本库。
Git的版本库里存了很多东西，其中最重要的就是称为stage（或者叫index）的暂存区，还有Git为我们自动创建的第一个分支master，以及指向master的一个指针叫HEAD。

我们把文件往Git版本库里添加的时候，是分两步执行的：
第一步是⽤用“git add”把文件添加进去，实际上就是把文件修改添加到暂存区；
第二步是⽤用“git commit”提交更改，实际上就是把暂存区的所有内容提交到当前分支。

小结
暂存区是Git非常重要的概念，弄明白了暂存区，就弄明白了Git的很多操作到底干了什么。
墨迹绘图
墨迹绘图
墨迹绘图
commit id （版本号）
墨迹绘图
回退版本
墨迹绘图
查看当前内容
墨迹绘图
墨迹绘图
￼
  
暂存区是Git非常重要的概念，弄明白了暂存区，就弄明白了Git的很多操作到底干了什么。

Git跟踪并管理的是修改，而非文件。

当执行git log时，退出按 q

未命名图片.png 计算机生成了可选文字:
第一次修改·>add一>第二次修改·>add一>commit

小结
现在，你又理解了Git是如何跟踪修改的，每次修改，如果不add到暂存区，那就不会加入到commit中。

git checkout -- file可以丢弃工作区的修改：
$ git checkout -- readme.txt

撤销修改
① 一种是readme.txt自修改后还没有被放到暂存区，现在，撤销修改就回到和版本库⼀一模一样的状态；
$ git checkout -- readme.txt
②一种是readme.txt已经添加到暂存区后，又作了修改，现在，撤销修改就回到添加到暂存区后的状态。（用命令git reset HEAD file可以把暂存区的修改撤销掉（unstage），重
新放回工作区）
$ git reset HEAD readme.txt
Unstaged changes after reset:
M readme.txt

小结
又到了小结时间。
场景1：当你改乱了工作区某个文件的内容，想直接丢弃工作区的修改时，用命令git checkout -- file。
场景2：当你不但改乱了工作区某个文件的内容，还添加到了暂存区时，想丢弃修改，分两步，第一步用命令git reset HEAD file，就回到了场景1，第二步按场景1操作。
场景3：已经提交了不合适的修改到版本库时，想要撤销本次提交，参考版本回退一节，不过前提是没有推送到远程库。

删除文件

直接在文件管理器中把没用的文件删了，或者用rm命令删了：
未命名图片.png p 1 -TMOG97M 
S git add test. txt 
p 1 um@DESKTOP -TMDG97M 
S git commit -m "add 
[master 173bef51 add 
earnglt (master 
MINGR64 -/learngit (master) 
txt" 
test. txt 
I file changed, I insertion(+) 
create mode 100644 test. txt 
pl MINGW64 -/learngit (master) 
S rm test. txt 
plum@DESKTOP-TMDG97M MINGW64 -nearngit (master) 
S git status 
n branch master 
Changes not staged for commit: 
(use "git add/rm le>... " 
to update what be committed) 
(use "git checkout " 
deleted: 
no changes added to commit (use 
to discard changes in working directory) 
"git add" 
an d /or 
"git commit -a") 
  
未命名图片.png p 1 -TMOG97M 
S git add test. txt 
p 1 um@DESKTOP -TMDG97M 
S git commit -m "add 
[master 173bef51 add 
earnglt (master 
MINGR64 -/learngit (master) 
txt" 
test. txt 
I file changed, I insertion(+) 
create mode 100644 test. txt 
pl MINGW64 -/learngit (master) 
S rm test. txt 
plum@DESKTOP-TMDG97M MINGW64 -nearngit (master) 
S git status 
n branch master 
Changes not staged for commit: 
(use "git add/rm le>... " 
to update what be committed) 
(use "git checkout " 
deleted: 
no changes added to commit (use 
to discard changes in working directory) 
"git add" 
an d /or 
"git commit -a") 

一是确实要从版本库中删除该文件，那就用命令git rm删掉，并且
commit：
$ git rm test.txt
rm 'test.txt'
$ git commit -m "remove test.txt"
[master d17efd8] remove test.txt
1 file changed, 1 deletion(-)
delete mode 100644 test.txt
现在，文件就从版本库中被删除了。
未命名图片.png plunvDÉSKTOP-TMDG97M MINGW64 
—/learngit (master) 
S git rm test.txt 
rm 'test.txt' 
.p1umg0ESKTop-TMDG97M MINGwS4 
—/learngit (master) 
git commit -m "remove test. txt 
[master 77967 dl] remove test. txt 
I file changed, deletion(— 
delete mode 100644 test. txt 
.p1urVOESkTOP-TMDG97M MINGw64 
—/learngit (master) 
'S git checkout test. txt 
error: pathspec 'test . txt' did not match any file(s) known to git. 

另一种情况是删错了，因为版本库里还有呢，所以可以很轻松地把误删的文件恢复到最新版本：
$ git checkout -- test.txt
git checkout其实是用版本库里的版本替换工作区的版本，无论工作区是修改还是删除，都可以“一键还原”。


小结
命令git rm用于删除一个文件。如果一个文件已经被提交到版本库，那么你永远不用担心误删，但是要小心，你只能恢复文件到最新版本，你会丢失最近一次提交后你修改的内容。

远程仓库

先有本地库，后有远程库的时候，如何关联远程库
Github——Git远程仓库
墨迹绘图
从工作区删除文件（此时工作区和版本库的版本不一致）
墨迹绘图
从版本库中删除文件
墨迹绘图
若误删，则可用版本库里的版本替换工作区的版本，无论工作区是修改还是删除，都可以“一键还原”
  
Github——Git远程仓库
第1步：创建SSH Key
$ ssh-keygen -t rsa -C "youremail@example.com"
未命名图片.png L? 
id rsa 
id rsa.pub 
2018 / 4 / 311 : 16 
2018 / 4 / 311 : 16 
PUB 文 件 
生成SSH Key的秘钥对，id_rsa是私钥（保密），id_rsa.pub是公钥（公开）
将SSH Key添加到Github中（证实身份）

添加远程库
1、在github上面创建一个新的仓库（Repository）
2、本地已经有一个Git仓库，Github上也有一个，将两个仓库进行远程同步





（此时，容易因为readme.txt发生冲突，所以最好在github上面不要生成readme.txt）
3、$ git push origin master
把本地master分支的最新修改推送至GitHub

小结
要关联一个远程库，使用命令git remote add origin git@server-name:path/repo-name.git；
关联后，使用命令git push -u origin master第一次推送master分支的所有内容；
此后，每次本地提交后，只要有必要，就可以使用命令git push origin master推送最新修改；
分布式版本系统的最大好处之一是在本地工作完全不需要考虑远程库的存在，也就是有没有联网都可以正常工作，而SVN在没有联网的时候是拒绝干活的！当有网络的时候，再把本地提交推送一下就完成了同步，真是太方便了！

从远程库克隆
先创建远程库，然后，从远程库克隆
1、在github上创建一个有README.md文件的仓库
2、用git clone 克隆一个本地库
未命名图片.png plumßDÉSKTOP-TMOG97M -/Desktop 
S git clone 
cloning into 'Daily_Learning' 
Warning: Permanently added the RSA host key for IP address '13.229.188.59' 
e list of known hosts. 
remote: counting objects: 3, done. 
remote: Total 3 (delta O), reused O (delta O), pack—reused O 
Receiving objects: 100% (3/3), done. 
to 
plumaDESKTOP-TMOG97M MING'*'64 
S cd Daily—Learning 
plumØDESKTOP-TMDG97M MING'*'64 
README . md 
— (Desktop 
—/Desktop/Dai ly_Learning (master) 

小结
墨迹绘图
两种方式：
是否有readme.txt文件（
①在github上新建一个仓库（有readme.txt文件），但是本地没有init操作（即还不是一个仓库）
echo "# Test" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin git@github.com:xinwenmei/Test.git
git push -u origin master


② 本地已经init了，即两个仓库进行同步
未命名图片.png ...or push an existing repository from the command line 
git remote add origin 
git push -u origin master 
墨迹绘图
  
小结
要克隆一个仓库，首先必须知道仓库的地址，然后使用git clone命令克隆。
Git支持多种协议，包括https，但通过ssh支持的原生git协议速度最快。

分支管理
创建与合并分支
每次提交，Git都把它们串成一条时间线，这条时间线就是一个分支。截止到目前，只有一条时间线，在Git里，这个分支叫主分支，即 master分支。HEAD严格来说不是指向提交，而是指向master，master才是指向提交的，所以，HEAD指向的就是当前分支。
未命名图片.png plunVOESKTOP-TMDG97M MINGW64 ly_Learning (master) 
S git checkout -b dev 
switched to a new branch 'dev' 
pluraOESKTOP-TMDG97M 
S git branch 
* dev 
mas ter 
plur00ESKTOP-TMDG97M MINGw64 
S git add readme. txt 
plur00ESKTOP-TMDG97M V11NGu64 
S git commit -m "branch test" 
(dev 2702099) branch test 
—/Oesktop/Dai ly—Learning (dev) 
I y_Learning (dev) 
I y_Learning (dev) 
I file changed, I insertion(+) 
create mode 100644 readme. txt 
plunOOESKTOP-TMDG97M MINGW64 ly_Learning (dev) 
S git checkout master 
switched to branch 'master' 
Your branch is up to date with 'origin/master' . 

未命名图片.png pl um@DESKTOP-TMDG97M MINGW64 
S git merge dev 
updating 29%219..2702099 
Fast-forward 
readme. txt I 
-/Desktop/Dai ly_Learning (master) 
I file changed, 1 insertion(+) 
create mode 100644 readme. txt 
plum@DESKTOP-TMOG97M MINGW64 -/Desktop/Dai1y_Learning (master) 
S git branch -d dev 
Deleted branch dev (was 2702099) . 
plum@DESKTop-TMDG97M MINGW64 -"Desktop/Dai1y_Learning (master) 
S git branch 

小结
Git鼓励大量使用分支：
查看分支：git branch
创建分支：git branch name
切换分支：git checkout name
创建+切换分支：git checkout -b name
合并某分支到当前分支：git merge name
删除分支：git branch -d name

解决冲突
墨迹绘图
git checkout命令加上-b参数表示创建并切换
   =  git branch dev
       git checkout dev

查看当前分支（*号是表示当前分支）
墨迹绘图
墨迹绘图
切换回master分支（提交的readme.txt在branch分支下面）
墨迹绘图
把dev分支的工作成果合并到master分支上
  
解决冲突

小结
当Git无法自动合并分支时，就必须首先解决冲突。解决冲突后，再提交，合并完成。
用git log --graph命令可以看到分支合并图。

分支管理策略

准备合并dev分支，请注意--no-ff参数，表示禁⽤用“Fast forward”：
$ git merge --no-ff -m "merge with no-ff" dev
合并要创建一个新的commit，所以加上-m参数，把commit描述写进去。

小结
Git分支⼗十分强大，在团队开发中应该充分应用。
合并分支时，加上--no-ff参数就可以用普通模式合并，合并后的历史有分支，能看出来曾经做过合并，而fast forward合并就看不出来曾经做过合并。

Bug分支
有了bug就需要修复，在Git中，由于分支是如此的强大，所以，每个bug都可以通过一个新的临时分支来修复，修复后，合并分支，然后将临时分支删除。

小结
修复bug时，我们会通过创建新的bug分支进行修复，然后合并，最后删除；
当手头工作没有完成时，先把工作现场git stash⼀一下，然后去修复bug，修复后，再git stash pop，回到工作现场。

Feature分支
添加一个新功能时，你肯定不希望因为一些实验性质的代码，把主分支搞乱了，所以，每添
加一个新功能，最好新建一个feature分支，在上面开发，完成后，合并，最后，删除该
feature分支。

小结
开发一个新feature，最好新建一个分支；
如果要丢弃一个没有被合并过的分支，可以通过git branch -D name强行删除。

多人协作

未命名图片.png pl MINGW64 /c/users/plum 
S cd learngit 
pl MING'"'64 /c/users/plum/learngit (master) 
S git remote 
or 'gin 
pl um@DESKTOP-TMOG97M MINGW64 /c/users/plum/learngit (master) 
S git remote -v 
gin gi nwenmei / learngit . git (fetch) 
ori gin gi nwenmei / learngit . git (push) 
pl MINGW64 /c/users/plum/learngit (master) 
S git push origin master 
Everything up-to-date 
墨迹绘图
查看远程库的信息
墨迹绘图
显示更详细的消息（推送权限才有push地址）
墨迹绘图
  
未命名图片.png pl MINGW64 /c/users/plum 
S cd learngit 
pl MING'"'64 /c/users/plum/learngit (master) 
S git remote 
or 'gin 
pl um@DESKTOP-TMOG97M MINGW64 /c/users/plum/learngit (master) 
S git remote -v 
gin gi nwenmei / learngit . git (fetch) 
ori gin gi nwenmei / learngit . git (push) 
pl MINGW64 /c/users/plum/learngit (master) 
S git push origin master 
Everything up-to-date 

因此，多人协作的工作模式通常是这样：
1. 首先，可以试图用git push origin branch-name推送自己的修改；
2. 如果推送失败，则因为远程分支比你的本地更新，需要先用git pull试图合并；
3. 如果合并有冲突，则解决冲突，并在本地提交；
4. 没有冲突或者解决掉冲突后，再用git push origin branch-name推送就能成功！
如果git pull提示“no tracking information”，则说明本地分支和远程分支的链接关系没
有创建，用命令git branch --set-upstream branch-name origin/branch-name。

小结
• 查看远程库信息，使用git remote -v；
• 本地新建的分支如果不推送到远程，对其他人就是不可见的；
• 从本地推送分支，使⽤用git push origin branch-name，如果推送失败，先用git pull抓
取远程的新提交；
• 在本地创建和远程分支对应的分支，使用git checkout -b branch-name origin/branchname，
本地和远程分支的名称最好⼀一致；
• 建立本地分支和远程分支的关联，使⽤用git branch --set-upstream branch-name
origin/branch-name；
• 从远程抓取分支，使用git pull，如果有冲突，要先处理冲突。

标签管理

墨迹绘图
把该分支（master）推送到远程库对应的远程分支上
