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，集中式和分布式版本控制系统有什么区别呢？

个人理解：

集中式版本控制系统有一个中央服务器，而完整代码只存在于这个中央服务器中，如果中央服务器坏了，那大家一起GG，而且断网的时候不支持commit、push、pull，比如说没有联网的时候你commit了10次，是找不到中间的修改的。

分布式版本控制系统虽然也有中央服务器，但是这个中央服务器只是方便大家修改和交换，每个人的个人服务器中，都有一个完整的代码，如果中央服务器坏了，只需要从任何一个人的本地clone一份就好了，断网的时候支持commit，每个修改都可以看到。不支持push、pull。

**再强调一次：git的“中央仓库”跟每个人的仓库是一摸一样的，仅仅用于方便交换修改**

创建仓库（版本库）repository

创建一个空仓库：

$mkdir yourRepositoryName

$cd yourRepositoryName

$pwd //显示当前目录

/g/17 vacation/testGit/yourRepositoryName

$git init //把这个目录变成Git可以管理的仓库

把要添加到仓库的文件放在目录下（或子目录下），在这里就是yourRepositoryName

$git add test.docx //告诉git把文件添加到仓库

$git commit -m "wrote a test file" //告诉git，把文件提交到仓库

-m后面输入的是本次提交的说明，最好是有意义的

$git status //可以让我们时刻掌握仓库当前的状态

$ git status

On branch master

Changes not staged for commit:

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

(use "git checkout -- <file>..." to discard changes in working directory)

modified: test.docx

no changes added to commit (use "git add" and/or "git commit -a")

//命令可以让我们时刻掌握仓库当前的状态，上面的命令告诉我们，test.docxt被修改过了，但还没有准备提交的修改。

$ git diff //查看difference,了解一下具体改了些什么

<C:/Users/ADMINI~1/AppData/Local/Temp/M0rAka\_test.docx> does not seem to be a docx file!

**diff --git a/test.docx b/test.docx**

**index e69de29..f82bfac 100644**

**--- a/test.docx**

**+++ b/test.docx**

@@ -0,0 +1 @@

+new

$ git add test.docx //修改后重新添加test文件到仓库

$ git commit -m"add new"//提交test文件

[master cde3ee1] add new

1 file changed, 0 insertions(+), 0 deletions(-)

$ git status //再看看仓库目前状态

On branch master

nothing to commit, working tree clean //非常好！Git告诉我们当前没有需要提交的修改，而且，工作目录是干净（working directory clean）的。

$git log //查看历史记录

commit cde3ee106dfab759ad9d553b10b7d136b616d593 (HEAD -> master)

Author: YuanJia <cnyuanjia@gmail.com>

Date: Wed Aug 23 13:35:48 2017 +0800

add new

commit 619da818856beeb7937334efa3e9f68c65e12e9c

Author: YuanJia <cnyuanjia@gmail.com>

Date: Wed Aug 23 12:57:37 2017 +0800

wrote a test file

git log命令显示从最近到最远的提交日志,最近一共是提交了两次，一次是创建了一个空文本，之后给文本加了一行字new

如果嫌输出信息太多，看得眼花缭乱的，可以试试加上--pretty=oneline参数

$ git log --pretty=oneline

cde3ee106dfab759ad9d553b10b7d136b616d593 (HEAD -> master) add new

619da818856beeb7937334efa3e9f68c65e12e9c wrote a test file

啊，一堆乱七八糟的是啥！

为什么commit id需要用这么一大串数字表示呢？因为Git是分布式的版本控制系统，后面我们还要研究多人在同一个版本库里工作，如果大家都用1，2，3……作为版本号，那肯定就冲突了。

现在想要把版本回退到一开始的 wrote a test file怎么做呢！

git reset命令！

首先，Git必须知道当前版本是哪个版本，在Git中，用HEAD表示当前版本，也就是最新的提交的add new版本，上一个版本就是HEAD^，上上一个版本就是HEAD^^，当然往上100个版本写100个^比较容易数不过来，所以写成HEAD~100。

$ git reset --hard Head~1 //这里写$git reset --hard Head^也行

HEAD is now at 619da81 wrote a test file

然后如果我后悔了，不想看到这个啥都没写的文本，还想回到别的版本，

$ git reflog//用来记录你的每一次命令

619da81 HEAD@{1}: reset: moving to Head~1

cde3ee1 (HEAD -> master) HEAD@{2}: commit: add new

619da81 HEAD@{3}: commit (initial): wrote a test file

就找到最新版本的ID啦

$ git reset --hard cde3

打开文件，我的new又回来了！ //Linux中有cat 命令显示文件内容，windows中没有（emmmmmmm..）

工作区和暂存区

（这篇文章思路跟着廖雪峰老师的博客走并借了几个图，尝试自己做一遍而已，并非原创）

工作区（Working Directory）

就是你在电脑里能看到的目录，比如我的testGit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。



第一步是用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。

因为我们创建Git版本库时，Git自动为我们创建了唯一一个master分支，所以，现在，git commit就是往master分支上提交更改。

你可以简单理解为，需要提交的文件修改通通放到暂存区，然后，一次性提交暂存区的所有修改。

现在我给test.docx加一行add

再新建一个文件example.docx

使用git status命令，看看现在的状况如何

$ git status

On branch master

Changes not staged for commit:

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

(use "git checkout -- <file>..." to discard changes in working directory)

modified: test.docx

Untracked files:

(use "git add <file>..." to include in what will be committed)

example.docx

no changes added to commit (use "git add" and/or "git commit -a")

Git非常清楚地告诉我们，test.docx被修改了，而example.docx还从来没有被添加过，所以它的状态是Untracked。

现在，使用两次命令git add，把test.docx和example.docx都添加后，用git status再查看一下：

$ git status

On branch master

Changes to be committed:

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

new file: example.docx

modified: test.docx

现在，暂存区的状态就变成这样了：



所以，git add命令实际上就是把要提交的所有修改放到暂存区（Stage），然后，执行git commit就可以一次性把暂存区（git commit不会管工作区的文件）的所有修改提交到分支。

$ git commit -m"add new file and change the test.docx"

[master f70c6a2] add new file and change the test.docx

2 files changed, 0 insertions(+), 0 deletions(-)

create mode 100644 example.docx

提交后再查看状态

$ git status

On branch master

nothing to commit, working tree clean

可以说是很干净了



git diff HEAD -- readme.txt命令可以查看工作区和版本库里面最新版本的区别：

我给我的test.docx加一行do，然后我使用一下这个命令

$ git diff HEAD -- test.docx

**diff --git a/test.docx b/test.docx**

**index 52c8306..f7134b2 100644**

**--- a/test.docx**

**+++ b/test.docx**

@@ -1,2 +1,3 @@

new

add

+do

如果我这时候直接commit而不先add会怎么样

$ git commit -m"a"

On branch master

Changes not staged for commit:

modified: test.docx

no changes added to commit

然后我不信邪的再次使用

$ git diff HEAD -- test.docx

**diff --git a/test.docx b/test.docx**

**index 52c8306..f7134b2 100644**

**--- a/test.docx**

**+++ b/test.docx**

@@ -1,2 +1,3 @@

new

add

+do

果然没有提交到仓库，因为：

**commit只管把暂存区的文档提交到仓库，而不会管工作区的事**

git diff #是工作区(work dict)和暂存区(stage)的比较

git diff --cached #是暂存区(stage)和分支(master)的比较

git diff HEAD #查看工作区和版本库里面最新版本的区别

现在我新增一个inStage.docx

目前工作区有一个有三行代码最后一行是do，并且do还没有提交到暂存区的test.docx，一个在最新版本库中的example.docx和一个我现在把他提交到暂存区的inStage.docx

然后我尝试一下git diff //是工作区(work dict)和暂存区(stage)的比较

目前情况（我认为）：

工作区：最新版本的test,example和inStage

暂存区：inStage

版本库：只有两行文字（并非最新版本）的test,最新版本的example

$ git diff

**diff --git a/test.docx b/test.docx**

**index 52c8306..f7134b2 100644**

**--- a/test.docx**

**+++ b/test.docx**

@@ -1,2 +1,3 @@

new

add

+do

因为工作区现在有三个文件，注意！！！

把工作区的文件提交到暂存区，是不会让工作区的文件不见的！他们还在！

但是情况还是有点不对啊，工作区和暂存区应该还有个inStage的差别

$ git diff HEAD//查看工作区和版本库里面最新版本的区别

**diff --git a/inStage.docx b/inStage.docx**

**new file mode 100644**

**index 0000000..1e9ef0c**

**--- /dev/null**

**+++ b/inStage.docx**

@@ -0,0 +1 @@

+in stage

**diff --git a/test.docx b/test.docx**

**index 52c8306..f7134b2 100644**

**--- a/test.docx**

**+++ b/test.docx**

@@ -1,2 +1,3 @@

new

add

+do

这个就非常合理！

仔细思考了一下，可能暂存区的内容commit提交到版本库，只是画了一个指针，而并没有确实删除暂存区的内容