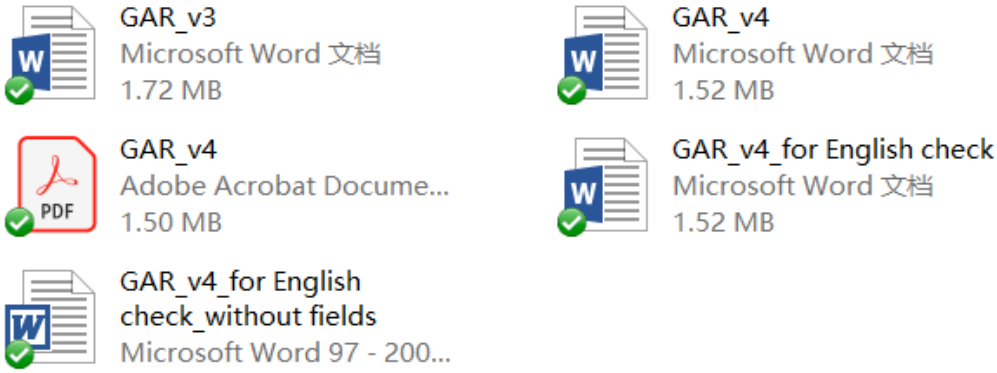
目录

1. Git简介

Git是目前世界上最先进的分布式版本控制系统

1.1 什么是版本控制系统?

我们从一个例子入手来理解版本控制系统，我最近在写一篇论文，每做一个更改（删除某一段），我都要保存成一个格外的版本，例如"GAR-V1", "GAR-V2", "GAR-V3",但是等过一段时候之后，我就经常忘记我到底做了什么修改，给我的科研进度造成了不少的困扰..



于是我想，如果有一个软件，不但能自动帮我记录每次文件的改动，还可以让同事协作编辑，这样就不用自己管理一堆类似的文件了，也不需要把文件传来传去。如果想查看某次改动，只需要在软件里瞄一眼就可以，岂不是很方便?

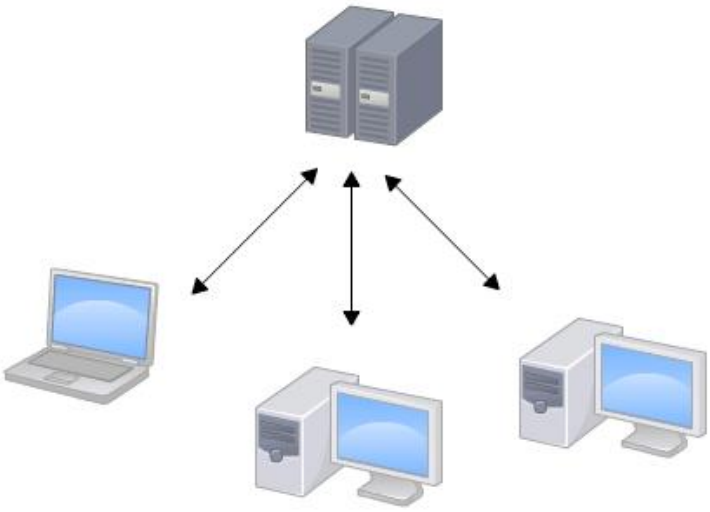
版本	文件名	用户	用户	用户
1	GAR.doc	张三	删除了Introduction最后一段	2018.12.2 13:32
2	GAR.doc	张三	增加了Literature Review的第一句话	2018.12.6 10:21
3	GAR.doc	李四	调整了Table1 的数据	2018.12.13 17:21

1.2. Git的历史

Linux诞生(1991)->Linux手动收集开源代码(2002)->BitKeeper版本控制系统托管Linux(2002-2005)->Andrew试图破解BitKeeper被发现，BitKeeper收回免费托管权(2005)->Linux自己写了Git，也就是Git的诞生(2005)

1.3. 什么是分布式？什么是集中式?

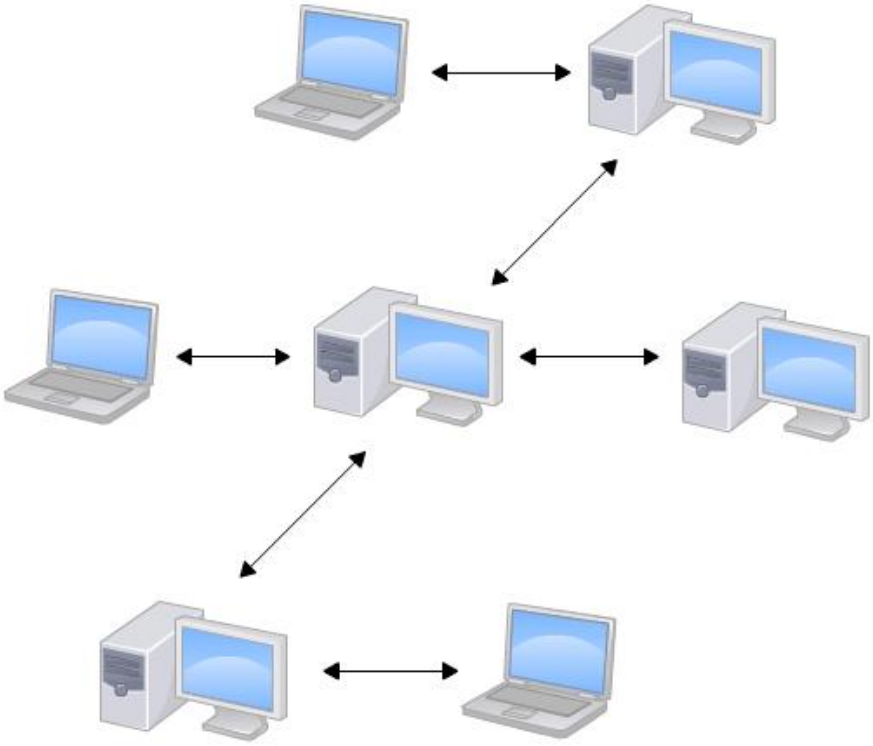
集中式:
集中式版本控制系统，版本库是集中存放在中央服务器的，而干活的时候，用的都是自己的电脑，所以要先从中央服务器取得最新的版本，然后开始干活，干完活了，再把自己的活推送给中央服务器。



集中式的缺点
集中式版本控制系统最大的毛病就是必须联网才能工作，如果在局域网内还好，带宽够大，速度够快，可如果在互联网上，遇到网速慢的话，可能提交一个10M的文件就需要5分钟

分布式:
分布式版本控制系统根本没有"中央服务器"，每个人的电脑上都是一个完整的版本库，这样，你工作的时候，就不需要联网了，因为版本库就在你自己的电脑上。既然每个人电脑上都有一个完整的版本库，那多个人如何协作呢？比方说你在自己电脑上改了文件A，

你的同事也在他的电脑上改了文件A，这时，你们俩之间只需把各自的修改推送给对方，就可以互相看到对方的修改了。



在实际使用分布式版本控制系统的时候，其实很少在两人之间的电脑上推送版本库的修改，因为可能你们俩不在一个局域网内，两台电脑互相访问不了，也可能今天你的同事病了，他的电脑压根没有开机。因此，分布式版本控制系统通常也有一台充当“中央服务器”的电脑，但这个服务器的作用仅仅是用来方便“交换”大家的修改，没有它大家也一样干活，只是交换修改不方便而已。

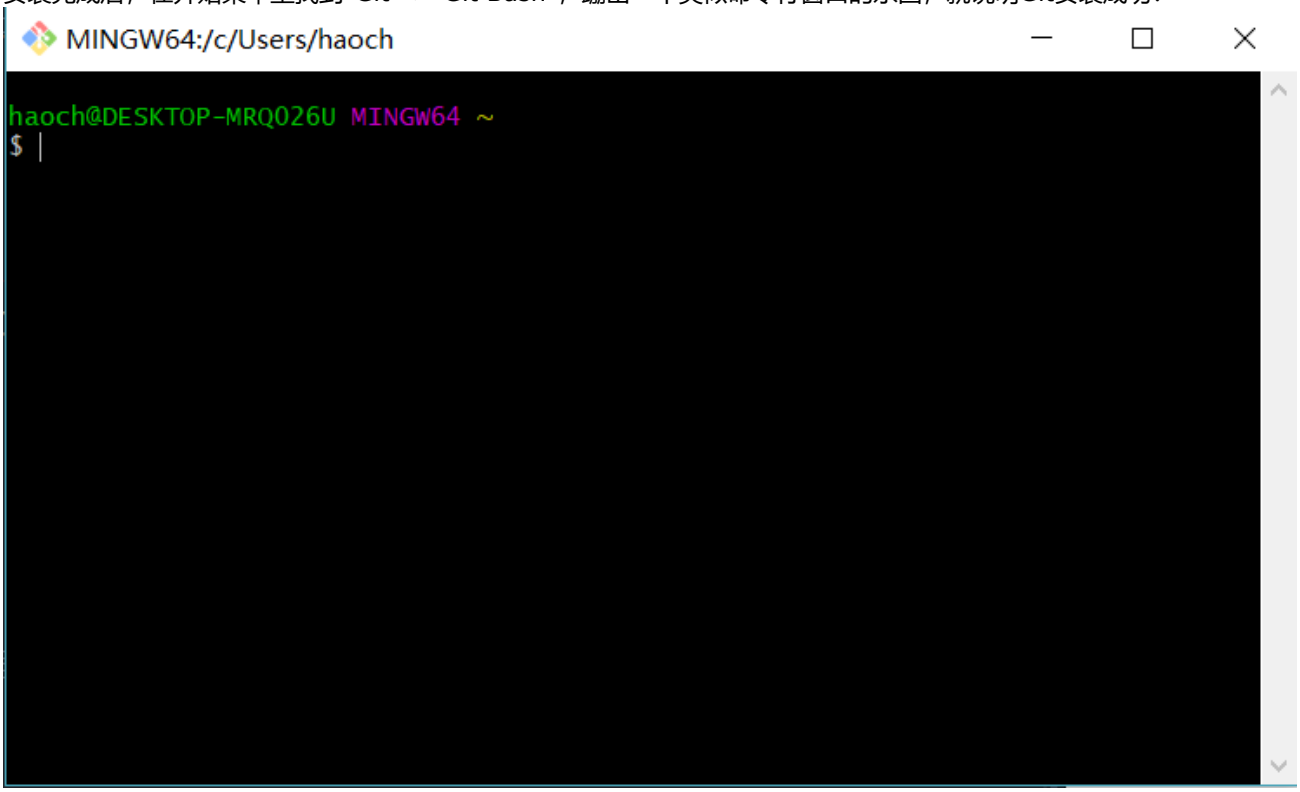
分布式优点

- 安全性高： 每个人电脑里都有完整的版本库，某一个人的电脑坏掉了不要紧，随便从其他人那里复制一个就可以了
- 免费

2. Git安装

先从Git官网下载安装程序,然后默认安装就可以了

安装完成后，在开始菜单里找到“Git”->“Git Bash”，蹦出一个类似命令行窗口的东西，就说明Git安装成功!



安装完成后，还需要最后一步设置，在命令行输入：

```
$ git config --global user.name "Your Name"
$ git config --global user.email "email@example.com"
```

3. 创建一个版本库

什么是版本库呢？版本库又名仓库，英文名repository，你可以简单理解成一个目录，这个目录里面的所有文件都可以被Git管理起来，每个文件的修改、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”。

创建一个版本库非常简单，首先，选择一个合适的地方，创建一个空目录,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 D:/git_test/.git/
```

瞬间 `Git` 就把仓库建好了，而且告诉你是一个空的仓库 `(empty Git repository)` ，细心的读者可以发现当前目录下多了一个 `.git` 的目录，这个目录是 `Git` 来跟踪管理版本库的，没事千万不要手动修改这个目录里面的文件，不然改乱了，就把 `Git` 仓库给破坏了。

---

4. Git的语法教程

4.1. 提交一个文件到版本库

先在刚刚创建好的repository下（D:/git_test）创建一个 `readme.txt` 文件，内容如下：

```
Git is a version control system.
Git is free software.
```

提交新添加的文件到仓库只需要两步： `添加(add) + 提交(commit)`

第一步： 用命令 `git add filename.txt` 告诉Git，把文件添加到repository：

```
$ git add readme.txt
```

第二步： 用命令 `git commit -m "description"` 告诉Git，把文件提交到仓库

```
$ git commit -m "write to readme"
[master (root-commit) f7f8050] write to readme
 1 file changed, 2 insertions(+)
 create mode 100644 readme.txt
```

`git commit` 命令执行成功后会告诉你， `1 file changed` 1个文件被改动（我们新添加的readme.txt文件）；
`2 insertions:` 插入了两行内容（readme.txt有两行内容）。

为什么Git添加文件需要add和commit两步?

因为 `commit` 可以一次提交很多文件，所以你可以多次 `add` 不同的文件

```
$ git add file1.txt
$ git add file2.txt file3.txt
$ git commit -m "add 3 files."
```

4.2.修改文件内容并提交

现在我们修改一下readme.txt文件内容为：

```
Git is a version control system.
Git is wonderful.
Git is free software.
```

现在我们可以通过 `git status` 来查看结果：

```
$ 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:   readme.txt

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

`git status` 命令可以让我们时刻掌握仓库当前的状态，上面的命令输出告诉我们，readme.txt被修改过了，但还没有准备提交的修改。

查看具体修改内容： 使用命令 `git diff`

```
$ git diff readme.txt
diff --git a/readme.txt b/readme.txt
index d8036c1..6e4b78e 100644
--- a/readme.txt
+++ b/readme.txt
@@ -1,2 +1,3 @@
-Git is a version control system.
+Git is wonderful.
-Git is free software.
\ No newline at end of file
```

上面显示我们新添加了一行文字

确定仓库状态和更改内容后，即可提交

第一步： add

```
$ git add readme.txt
```

查看repository状态：

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   readme.txt
```

上面告诉我们，将要提交的修改包括 `readme.txt`

第二步：commit

```
$ git commit -m "add new line"
[master 4fb84da] add new line
 1 file changed, 1 insertion(+)
```

查看repository状态：

```
$ git status
On branch master
```

```
nothing to commit, working tree clean
```

4.3. 版本回退

(1) 回退前先多提交几个版本

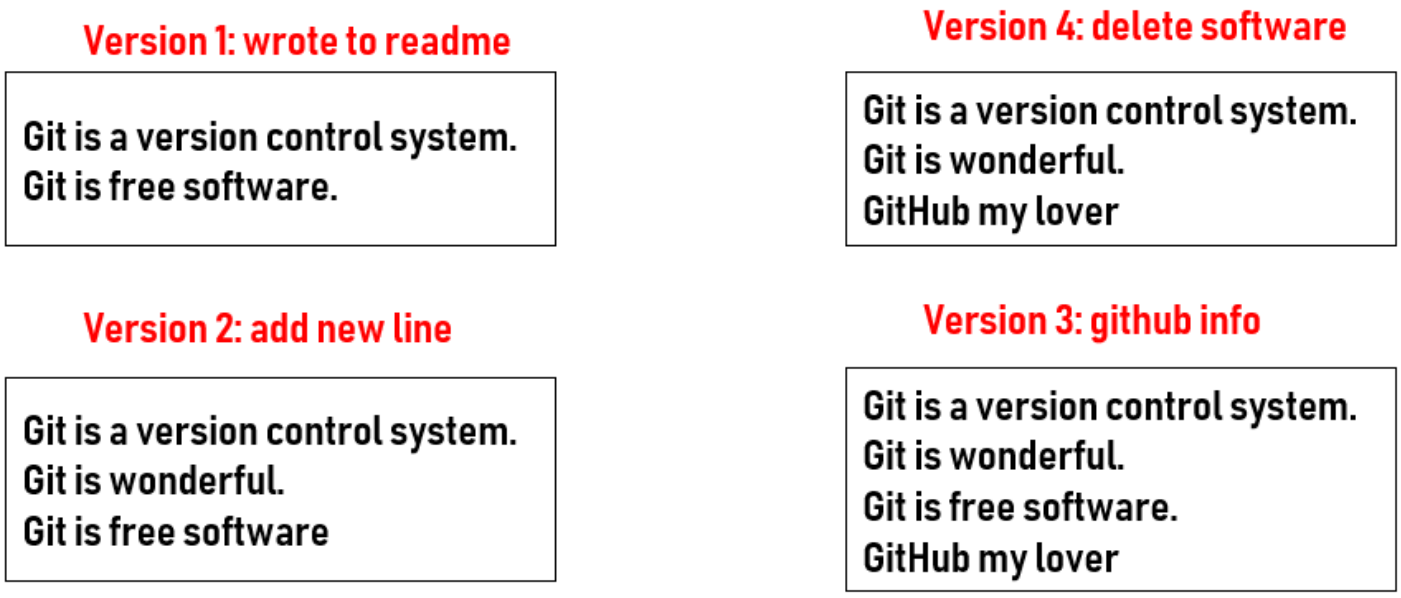
我们先重复修改几次文本并提交

```
Git is a version control system.
Git is wonderful.
GitHub my lover.
```

然后提交:

```
$ git commit -m "delete software"
[master 1094adb] delete software
 1 file changed, 1 insertion(+), 1 deletion(-)
```

现在让我们来想一下到底有几个版本被提交到仓库中了:



但是我们在实际工作中，无法记住这么多版本，我们可以通过命令 `git log` 来查看历史记录

```
$ git log
commit 5103166639e21fa3c8884f890ec53c0c4541f6d4 (HEAD -> master)
Author: haoch <haochen273@gmail.com>
Date:   Thu Jan 3 13:19:54 2019 +0900

    delete software

commit 9fbb43595b5aee1773fdd2db42427d9af9275db0
Author: haoch <haochen273@gmail.com>
Date:   Thu Jan 3 13:19:28 2019 +0900

    github infor

commit 4fb84da7882c74cc6dbd0b22ee64d8cf366e6a64
Author: haoch <haochen273@gmail.com>
Date:   Thu Jan 3 13:15:00 2019 +0900

    add new line

commit f7f80507a255a347a92117ff3bb75ddca837b91a
Author: haoch <haochen273@gmail.com>
Date:   Thu Jan 3 12:25:06 2019 +0900

    write to readme
```

我们也可以通过另外一个命令来简化历史记录的显示 `git log --pretty=oneline`

```
$ git log --pretty=oneline
5103166639e21fa3c8884f890ec53c0c4541f6d4 (HEAD -> master) delete software
9fbb43595b5aee1773fdd2db42427d9af9275db0 github infor
4fb84da7882c74cc6dbd0b22ee64d8cf366e6a64 add new line
f7f80507a255a347a92117ff3bb75ddca837b91a write to readme
```

(2). 版本回退到历史版本

现在-》过去

在Git中， `HEAD` 表示当前版本, 'HEAD^'表示上一个版本, 'HEAD^^'表示上上一个版本， `HEAD~10` 表示第前10个版本

现在，我们需要把当前版本 `delete software` 回退到 `github infor`

```
$ git reset --hard HEAD^
HEAD is now at 9fbb435 github infor
```

我们在来看看历史记录 `git log`

```
$ git log
commit 9fbb43595b5aee1773fdd2db42427d9af9275db0 (HEAD -> master)
Author: haoch <haochen273@gmail.com>
Date:   Thu Jan 3 13:19:28 2019 +0900

    github infor

commit 4fb84da7882c74cc6dbd0b22ee64d8cf366e6a64
Author: haoch <haochen273@gmail.com>
Date:   Thu Jan 3 13:15:00 2019 +0900

    add new line

commit f7f80507a255a347a92117ff3bb75ddca837b91a
Author: haoch <haochen273@gmail.com>
Date:   Thu Jan 3 12:25:06 2019 +0900

    write to readme
```

过去-》现在

其中已经没有了版本号 `delete software` ,好比你从21世纪坐时光穿梭机来到了19世纪，想再回去已经回不去了，肿么办?

办法其实还是有的，只要上面的命令行窗口还没有被关掉，你就可以顺着往上找啊找啊，找到那个 `delete software` 的commit id是510316...，于是就可以指定回到未来的某个版本:

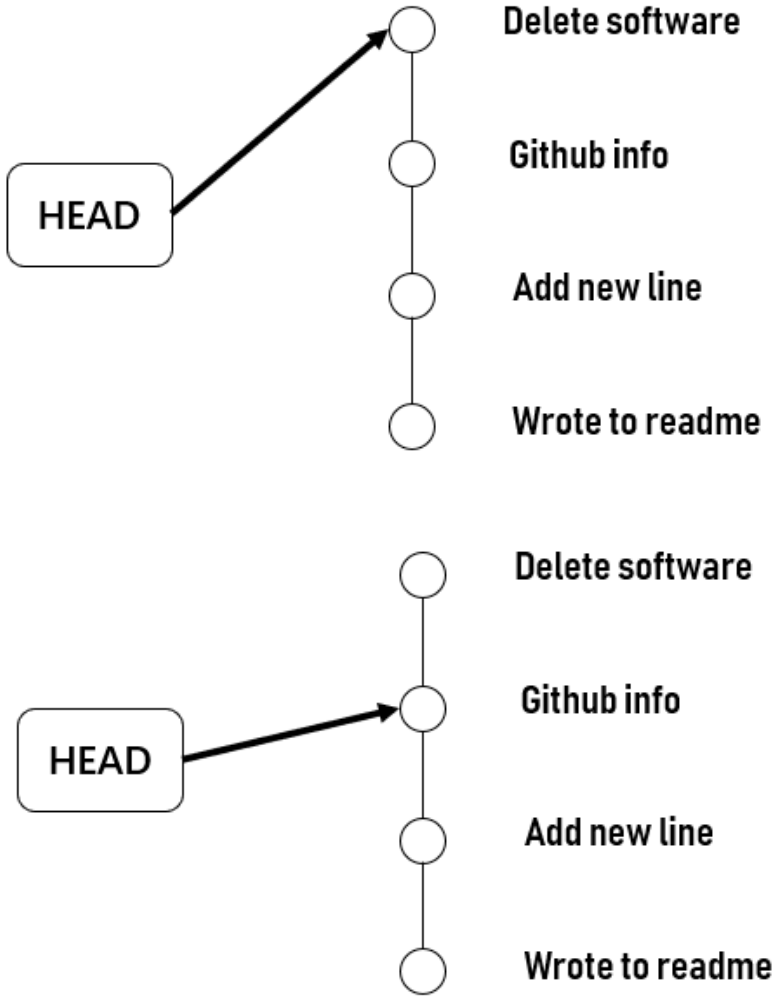
```
$ git reset --hard 510316
HEAD is now at 5103166 delete software
```

当你把电脑也关了，第二天起来想回到 `delete software` 版本怎么办？可以使用 `git reflog` 来查看每个提交的 `commit id` 在用以上的方法即可

```
$ git reflog
5103166 (HEAD -> master) HEAD@{0}: reset: moving to 510316
9fbb435 HEAD@{1}: reset: moving to HEAD^
5103166 (HEAD -> master) HEAD@{2}: commit: delete software
9fbb435 HEAD@{3}: commit: github infor
4fb84da HEAD@{4}: commit: add new line
f7f8050 HEAD@{5}: commit (initial): write to readme
```

HEAD的作用

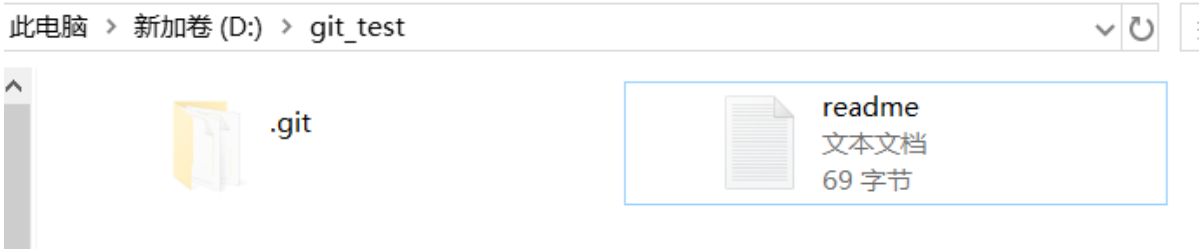
相当于C语言中的指针一样



4.4. 工作区和暂存区

（1）工作区

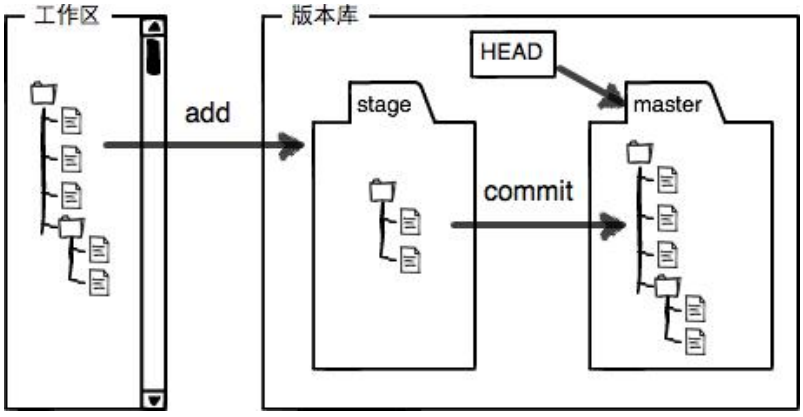
就是你在电脑里能看到的目录， 例如我的 `git_test` 文件夹



（2）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` 。



前面讲了我们把文件往Git版本库里添加的时候，是分两步执行的：

- `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；
- `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。

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。

（3）实例演示

让我们通过一个例子来演示工作区和版本库的作用

先对 `readme.txt` 进行一些更改为加一段文字I come from China：

```
Git is a version control system.
Git is wonderful.
GitHub my lover.
I come from China
```

然后在工作区新增一个名字为 `license.txt` 文件，内容随意

用 `git status` 查看状态

```
$ 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:   readme.txt

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

        license.txt

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

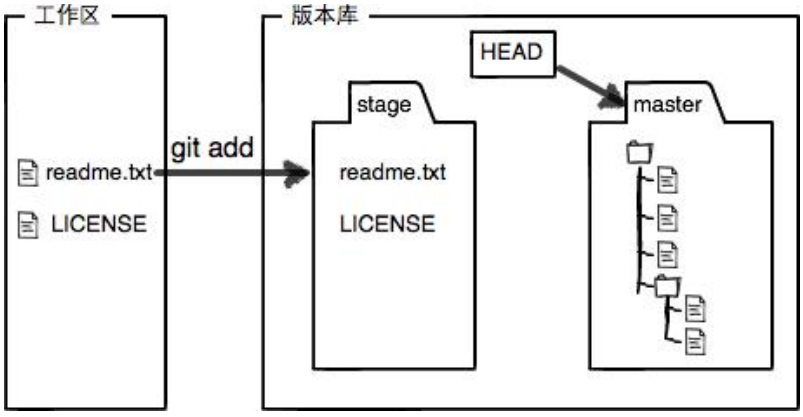
上面告诉我们： `readme.txt` 被修改了，但是 `licese.txt` 因为没有被添加，所以是无法跟踪的untracked

我们通过 `add` 将两个文件都添加后再查看状态：

```
$ git add readme.txt license.txt
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   license.txt
        modified:   readme.txt
```

现在暂存区的状态就变成这样了：



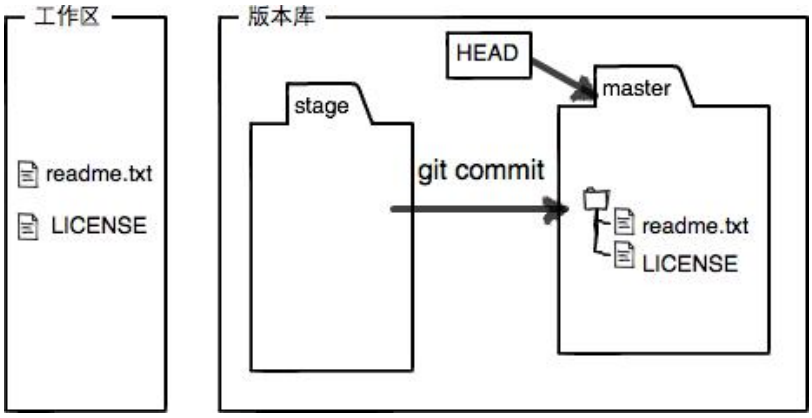
`git add` 就是把所有的修改都放到暂存区中，然后执行一次 `git commit` 就可以一次性把暂存区的所有修改提交到分支

```
$ git commit -m "how it works"
[master 45700b2] how it works
 2 files changed, 3 insertions(+), 1 deletion(-)
 create mode 100644 license.txt
```

一旦提交后，如果你又没有对工作区做任何修改，那么工作区就是"干净"的：

```
$ git status
On branch master
nothing to commit, working tree clean
```

现在暂存区就是这样的：



4.5. 管理修改

修改 `readme.txt` 内容为添加一行：

```
I love China
```

通过 `git add` 添加

再次修改修改 `readme.txt` 内容为添加一行：

```
Do you like China?
```

再通过 `git commit` 提交，在查看状态

```
$ 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:   readme.txt

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

让我们来回顾一下：

第一次修改 -> `git add` -> 第二次修改 -> `git commit`

Git管理的是修改，当你用 `git add` 命令后，在工作区的第一次修改被放入暂存区，准备提交，但是，在工作区的第二次修改并没有放入暂存区，所以， `git commit` 只负责把暂存区的修改提交了，也就是第一次的修改被提交了，第二次的修改不会被提交。

提交后，用 `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：

```
$ git diff HEAD -- readme.txt
diff --git a/readme.txt b/readme.txt
index 812e617..d5258f9 100644
--- a/readme.txt
+++ b/readme.txt
@@ -2,4 +2,5 @@ Git is a version control system.
 Git is wonderful.
 GitHub my lover.
 I come from China
-I love China
\ No newline at end of file
+I love China
+Do you love china?
\ No newline at end of file
```

第二次修改确实没有被提交。

如何提交第二次修改

第一次修改 -> `git add` -> 第二次修改 -> `git add` -> `git commit`

4.6. 撤销修改

比如说你的文件里面加了一段不好的话，你想删除他的方法：

```
Git is a version control system.
Git is wonderful.
GitHub my lover.
I come from China
I love China
Do you love china?
My stupid techer
```

```
haoch@DESKTOP-MRQO26U MINGW64 /d/git_test (master)
$ cat readme.txt
Git is a version control system.
Git is wonderful.
GitHub my lover.
I come from China
I love China
My stupid teacher
haoch@DESKTOP-MRQO26U MINGW64 /d/git_test (master)
$ git add readme.txt
```

此时，你可以使用以下命令来丢弃工作区的修改：

```
$ git checkout -- readme.txt
```

命令 `git checkout -- readme.txt` 意思就是，把 `readme.txt` 文件在工作区的修改全部撤销，这里有两种情况：

- 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模一样的状态；
- 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。

总之，就是让这个文件回到最近一次 `git commit` 或 `git add` 时的状态。

1）情况1：未添加到暂存区

现在再来看 `readme.txt` 的内容：

```
Git is a version control system.
Git is wonderful.
GitHub my lover.
I come from China
I love China
```

2）情况2：添加到暂存区

加入你在 `commit` 之前发现了这个问题，可以用以下命令撤销暂存区的修改

```
git reset HEAD <file>
```

就是把暂存区的修改退回到工作区

然后在工作区中使用情况1中的命令来删除修改

```
$ git checkout -- readme.txt
```

再次查看时候，已经回到了原来的版本了

```
Git is a version control system.
Git is wonderful.
GitHub my lover.
I come from China
I love China
```

4.7. 文件删除和恢复

Git是管理修改的，删除也是一种修改，也可以被管理

在工作区新建一个文件 `text.txt` 并且添加提交

```
$ git add text.txt

$ git commit -m "add text.txt"
[master b84166e] add text.txt
 1 file changed, 1 insertion(+)
 create mode 100644 text.txt
```

一般情况下，你通常直接在文件管理器中直接把文件删除了

```
$ rm text.txt
```

Git知道你删除了文件，因此，工作区和版本库就不一致了， `git status` 命令会立刻告诉你哪些文件被删除了：

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        deleted:    text.txt

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

1)情况1：真的要删除文件： `git rm` -> `git commit`

```
$ git rm text.txt
rm 'text.txt'

$ git commit -m "remove text.txt"
[master 1f108b4] remove text.txt
 1 file changed, 1 deletion(-)
 delete mode 100644 text.txt
```

2)情况2：你删错了，想恢复这个文件： `git checkout -- text.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”。