1. 关于版本控制

一种记录一个或若干文件内容变化，以便将来查阅特定版本修订情况的系统。 在我们所展示的例子中，我们对保存着软件源代码的文件作版本控制，但实际上，你可以对任何类型的文件进行版本控制。

> 有了版本控制，你就可以将某个文件回溯到之前的状态，甚至将整个项目都回退到过去某个时间点的状态，你可以比较文件的变化细节，查出最后是谁修改了哪个地方，从而找出导致怪异问题出现的原因，又是谁在何时报告了某个功能缺陷等等。 使用版本控制系统通常还意味着，就算你乱来一气把整个项目中的文件改的改删的删，你也照样可以轻松恢复到原先的样子。 但额外增加的工作量却微乎其微。

集中化的版本控制系统

这类系统，诸如 CVS、Subversion 以及 Perforce 等，都有一个单一的集中管理的服务器，保存所有文件的修订版本，而协同工作的人们都通过客户端连到这台服务器，取出最新的文件或者提交更新。

> 机械老哥们使用的PDM模型管理系统，需要一台部署在实验室的服务器，需要修改模型的时候从服务器中检出模型进行修改，修改完再提交到服务器，这个过程中其他人无法对模型进行修改；同时，所有模型都存储在服务器硬盘上，如果服务器宕机，所有模型都无法打开或者修改，如果服务器损坏，带来的损失也是无法想象的；

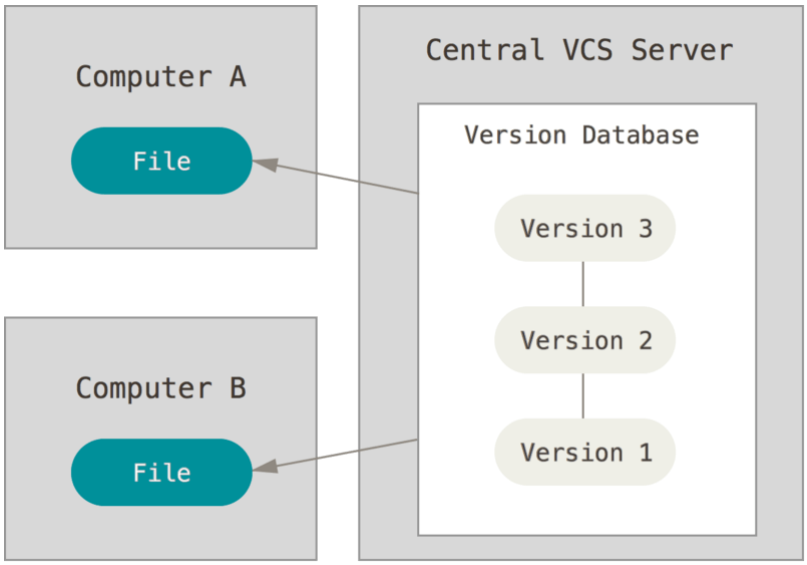

图1.1 集中化版本控制系统VCS

分布式的版本控制系统

这类系统中，像 Git、Mercurial、Bazaar 以及 Darcs 等，客户端并不只提取最新版本的文件快照，而是把代码仓库完整地镜像下来。 这么一来，任何一处协同工作用的服务器发生故障，事后都可以用任何一个镜像出来的本地仓库恢复。 因为每一次的克隆操作，实际上都是一次对代码仓库的完整备份。

> 更进一步，许多这类系统都可以指定和若干不同的远端代码仓库进行交互。籍此，你就可以在同一个项目中，分别和不同工作小组的人相互协作。 你可以根据需要设定不同的协作流程，比如层次模型式的工作流，而这在以前的集中式系统中是无法实现的。

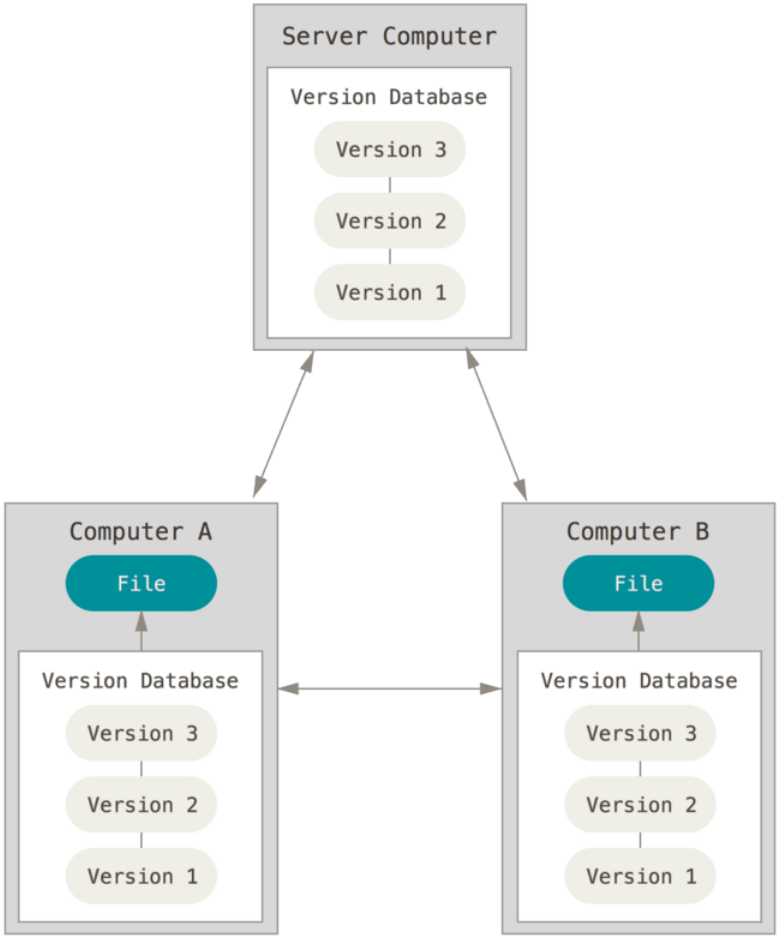

图1.2 分布式版本控制系统GIT

2. Git 基础

直接记录快照，而非差异比较

Git 更像是把数据看作是对小型文件系统的一组快照。 每次你提交更新，或在 Git 中保存项目状态时，它主要对当时的全部文件制作一个快照并保存这个快照的索引。 为了高效，如果文件没有修改，Git 不再重新存储该文件，而是只保留一个链接指向之前存储的文件。 Git 对待数据更像是一个 **快照流**。

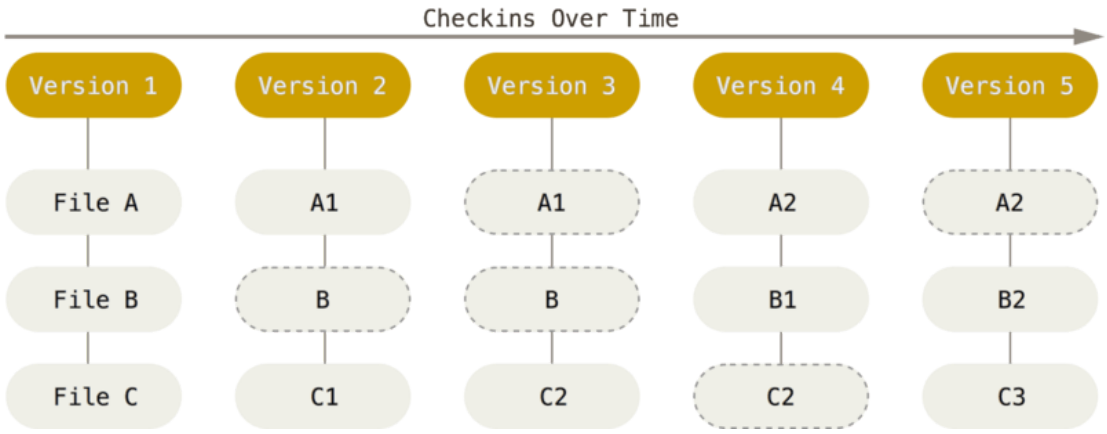

图2.1 Git 数据库结构

近乎所有操作都是本地执行

在 Git 中的绝大多数操作都只需要访问本地文件和资源，一般不需要来自网络上其它计算机的信息。

> 举个例子，要浏览项目的历史，Git 不需外连到服务器去获取历史，然后再显示出来——它只需直接从本地数据库中读取。 你能立即看到项目历史。 如果你想查看当前版本与一个月前的版本之间引入的修改，Git 会查找到一个月前的文件做一次本地的差异计算，而不是由远程服务器处理或从远程服务器拉回旧版本文件再来本地处理。

Git 保证完整性

Git 中所有数据在存储前都计算校验和，然后以校验和来引用。 这意味着不可能在 Git 不知情时更改任何文件内容或目录内容。
Git 用以计算校验和的机制叫做 SHA-1 散列（hash，哈希）。 这是一个由 40 个十六进制字符（0-9 和 a-f）组成的字符串，基于 Git 中文件的内容或目录结构计算出来。

Git 一般只添加数据

你执行的 Git 操作，几乎只往 Git 数据库中增加数据。 很难让 Git 执行任何不可逆操作，或者让它以任何方式清除数据。一旦你提交快照到 Git 中，就难以再丢失数据，特别是如果你定期的推送数据库到其它仓库的话。

Git 的三种状态（重点）

Git 有三种状态，你的文件可能处于其中之一：**已提交（committed）**、**已修改（modified）** 和 **已暂存（staged）**。

> 已提交表示数据已经安全的保存在本地数据库中。
> 已修改表示修改了文件，但还没保存到数据库中。
> 已暂存表示对一个已修改文件的当前版本做了标记，使之包含在下次提交的快照中。

由此引入 Git 项目的三个工作区域的概念：**工作目录**、**暂存区域** 以及 **Git 仓库**。

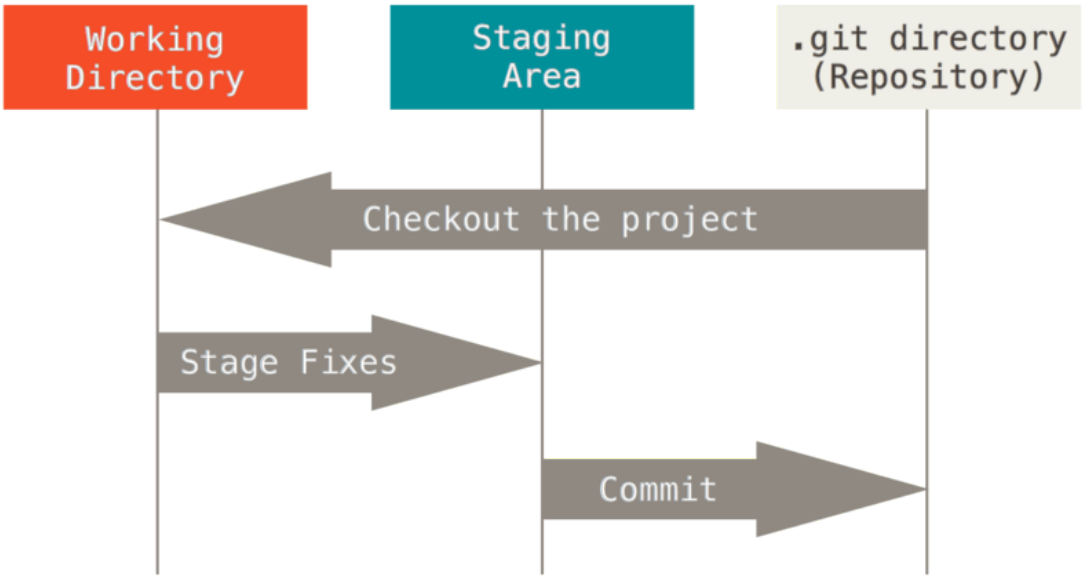

图2.2 Git 三个区域

基本的Git工作流程如下：

- 在工作目录中修改文件。
- 暂存文件，将文件的快照放入暂存区域。
- 提交更新，找到暂存区域的文件，将快照永久性存储到 Git 仓库目录。

3. 准备阶段

安装 Git

下载：https://git-scm.com/download/win

- 选择64位版本
- q群里也有安装包

安装：

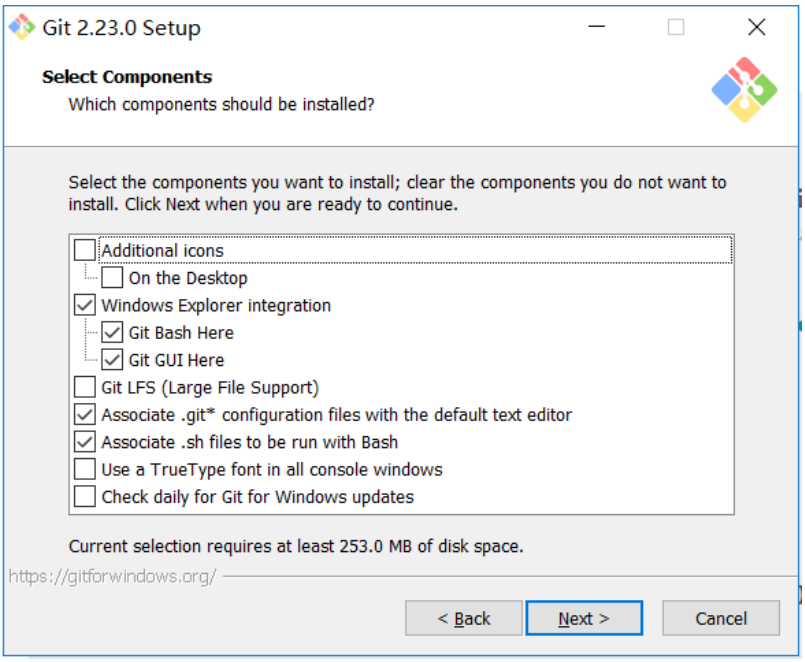

图3.1 默认勾选即可

选择 Git 的默认编辑器，推荐 Linux 风格的 Vim 编辑器，另外如果有安装其他编辑器比如 Notepad++ 也可以；

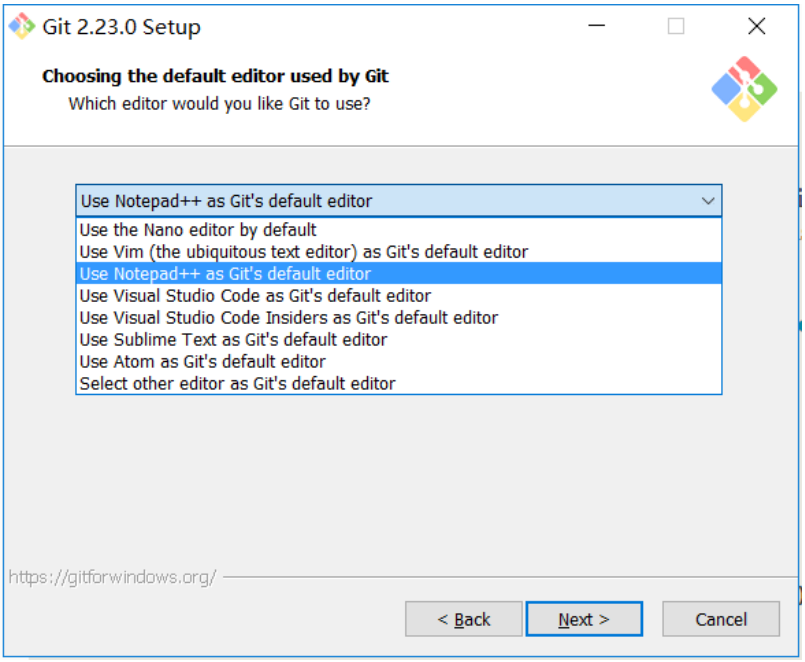

图3.2 选择默认编辑器

命令行

学习 Git 的时候，建议大家从命令行操作学起，只有在命令行下进行操作才能执行所有 Git 指令，在操作其他 Git 的 GUI 软件时也不会遇到太大的困难；

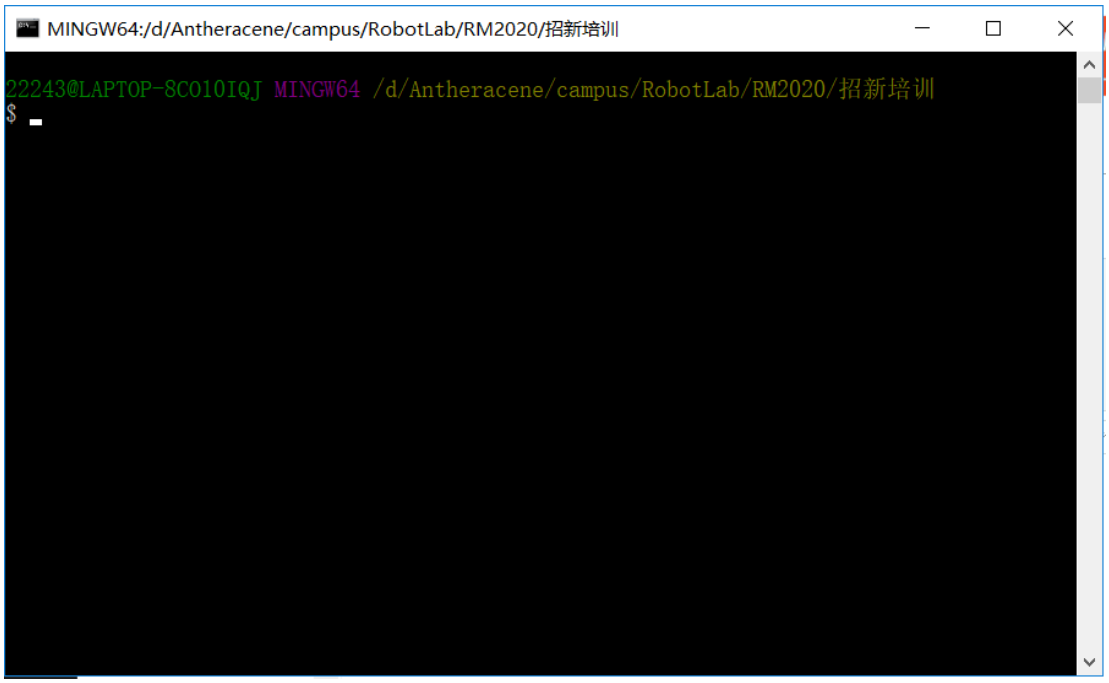

图3.3 Git Bash 命令行窗口

初次运行 Git 前的配置

需要做几件事来定制你的 Git 环境。 每台计算机上只需要配置一次，程序升级时会保留配置信息。 你可以在任何时候再次通过运行命令来修改它们。

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

这里使用了 `--global` 选项，以上两条命令只运行一次，之后 Git 会一直使用这些信息， 当你想针对特定项目使用不同的用户名称与邮件地址时，可以在那个项目目录下运行没有 `--global` 选项的命令来配置。

你可以使用 `git config --list` 命令来列出当前 Git 使用的配置信息：

```
$ git config --list
core.symlinks=true
core.autocrlf=input
core.fscache=true
user.email=a742674806@163.com
user.name=Anthracene
...
```

4. 初步使用

获取帮助

Git 有几种获取帮助的方法：

```
$ git [bref] --help
$ git help [bref]
```

运行以上命令，系统会打开该命令详细介绍的 html 页面：

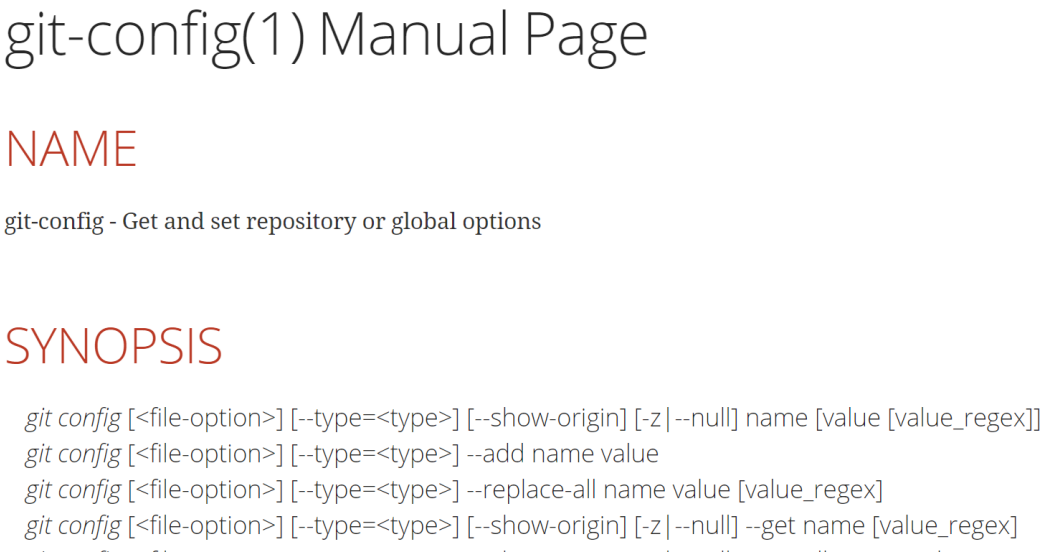

图4.1 Git Help html界面

获取 Git 仓库

a) 在现有目录中新建仓库

在需要新建仓库的根目录下右键，在右键菜单中打开 Git Bash 命令行窗口，或者直接在 Git 命令行窗口中进入仓库的根目录；

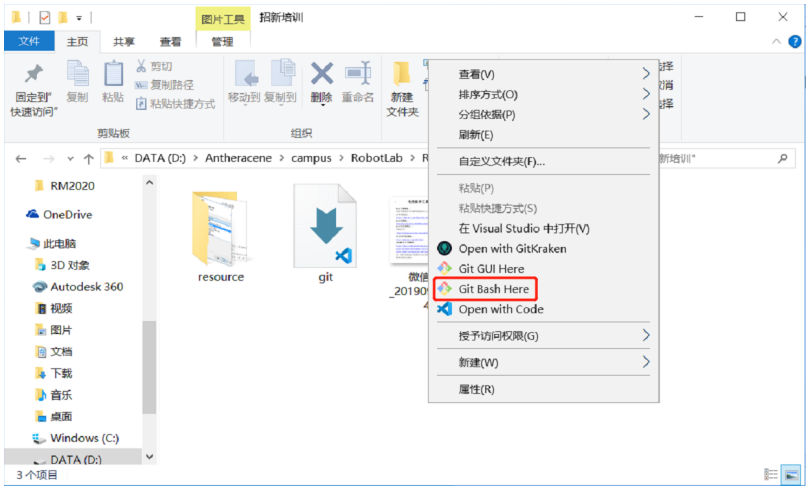

图4.2 右键菜单

如果使用命令行的 cd 指令，需要注意在命令行中地址是 Linux 风格的，不能直接从windows资源管理器里直接复制，需要把分隔符“\”更换为“/”，然后把“D:”更换为“/d”;

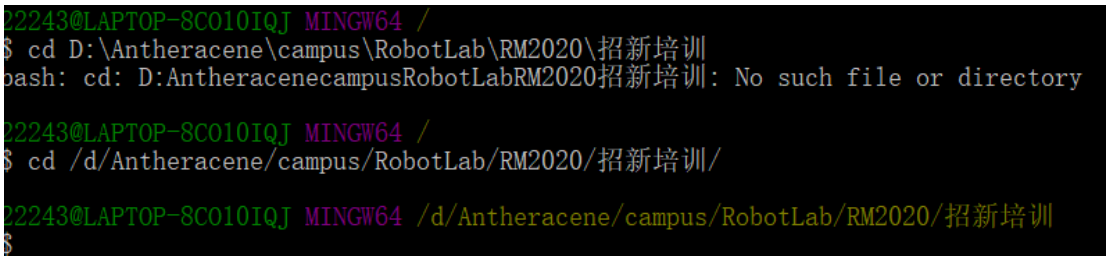

图4.3 切换目录

进入目录后，使用 `init` 指令来新建 Git 仓库：

```
$ git init
```

b) 克隆现有的仓库

使用 `clone [url]` 指令来克隆已有的 Git 仓库：

```
$ git clone https://gitee.com/Anthracene/mnist_font_recognition.git
```

为了与他人进行协作，我们会把本地仓库储存一个镜像在云端服务器，比如 github，gitee 和实验室自己的 Git 服务器，在服务器里的仓库称为远程仓库，本地修改的提交需要定期推送到远程仓库中来同步与他人的协作；

记录每次更新到仓库

养成习惯，在每次修改代码之前，你需要先了解一下当前 Git 仓库的状态，是否存在修改过未提交的文件，在命令行操作中，使用 `status` 指令来查看当前仓库的状态：

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

当你创建或修改了一个文件，使用 `add` 指令来跟踪他：

```
$ git add README.md
```

再运行 `git status` 指令，我们可以看到仓库状态的变化：

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

    new file:   README.md
```

这个时候，该文件已经被暂存到暂存区中；
当你有很多目录，很多文件，你可以在 `add` 指令后面附上目录路径作为参数，Git 会自动地递归地将目录下所有文件添加到仓库中；

忽略文件

在本地文件夹中，你可能有一些文件不想提交到仓库中，比如一些编译生成的二进制文件 `*.o` ，为了方便我们提交需要提交的文件，我们将这些没有必要添加到仓库中的文件记录起来；
在本仓库的根目录中，创建一个文件 `.gitignore` ：

```
$ touch .gitignore
```

使用记事本编辑文件，添加需要忽略的文件或路径到该文件中；

```
$ notepad .gitignore
```

> 【规范】
> 在我们的嵌入式软件工程中，存在一些不能提交到仓库的文件，比如编译的产物，本地用户的工程文件，以名为 MainControl 的工程为例，.gitignore文件需要包括：
> MainControl/MainControl
> MainControl/Project/*.bak
> MainControl/Project*.dep
> MainControl/Project/project.uvgui.*
> MainControl/Project/Obj/*
> MainControl/Project/List/*
> MainControl/Project/JLinkLog.tx
> *.rar
> *.o

其他需要忽略的文件类型，参考 https://github.com/github/gitignore 中关于 C 和 C++ 的.gitignore 模板。

查看已暂存和未暂存的修改

使用 `git diff` 指令查看尚未暂存的文件更新了哪些部分；

```
$ git diff
diff --git a/CONTRIBUTING.md b/CONTRIBUTING.md
index 8ebb991..643e24f 100644
--- a/CONTRIBUTING.md
+++ b/CONTRIBUTING.md
@@ -65,7 +65,8 @@ branch directly, things can get messy.
 Please include a nice description of your changes when you submit your PR;
 if we have to read the whole diff to figure out why you're contributing
 in the first place, you're less likely to get feedback and have your change
-merged in.
+merged in. Also, split your changes into comprehensive chunks if your patch is
+longer than a dozen lines.

 If you are starting to work on a particular area, feel free to submit a PR
 that highlights your work in progress (and note in the PR title that it's
```

使用带 `--staged` 参数的 `git diff` 指令来查看暂存的文件的改动

```
$ git diff --staged
diff --git a/README b/README
new file mode 100644
index 0000000..03902a1
--- /dev/null
+++ b/README
@@ -0,0 +1 @@
+My Project
```

提交更新

推荐使用 `-m` 参数（message） 来对本次提交进行简短的描述

```
$ git commit -m "[Feat] 增加板间通信停止PDO的接口"
```

每次提交时要好好斟酌这次提交的主要更改在哪里，在提交的简报中尽量描述清楚，以便以后进行版本控制；

> 【规范】
> 根据提交的版本更改类型选择以下相对应标识符在 commit 的 message 中注明:
> Feat: 增加新功能
> Fix: 修复Bug
> Docs: 编写文档
> Refactor: 代码重构
> Test: 增加测试代码

查看提交历史

使用 `git log` 命令来查看提交历史

```
$ git log
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon
Date:   Mon Mar 17 21:52:11 2008 -0700

    changed the version number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon
Date:   Sat Mar 15 16:40:33 2008 -0700

    removed unnecessary test

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon
Date:   Sat Mar 15 10:31:28 2008 -0700

    first commit
```

当进行代码审查，或者快速浏览某个搭档提交的 commit 所带来的变化的时候，可以使用 `-p` 参数来显示历史提交之间的内容差异；
如果想要查看大量提交信息，请加上 `--pretty=oneline` 作为参数：

```
$ git log --pretty=oneline
8efa34ead5174ccfe0b424d8a63e39ab314a34b1 (HEAD -> dev_rune) Merge branch 'dev_rune' of https://w
277a2b375f16a506bb04f07b62ec247f095ffc94 未知
56afe9c6e72a06aba39dd7d3fc220ac6dff797a0 (origin/dev_rune) 添加readme
1f45cf680c93b6ebcecf499b6d0be7721fd34a72 底盘和主控代码合并
1de8b43ef52827c295a8595e985be81948430215 修复了近距离打基地模式热量累计过快的bug
2947bec99610fc3e12a26e7e1558dd1e52ae6262 上场代码 打基地模式待测试
7aa9bdc14d60dfd3ddf03c3ea4c4995de74fd075 (origin/dev_chassis) 修改了开电容的时候的限幅
a6e881cb2f8c5c38997bcc43bf02c7a37e8b0400 测试可用的底盘版本，电机输出如果给的过大会烧坏升压板
50a304cde72f8aa44dc0b3bedb3de4885b63bc85 电容终于可以用了
4597fd06aa4aa4201a36f2ace42ec9a8041dedaf 神符可以自动打
38326ead8cf18ab146dbaab6dd73d95e82f8b6f2 去掉了裁判系统CRC校验，解决卡死问题，修改了电容控制部分，增
b2e7ecabf9e9a54d4774405f265849ca51457551 底盘响应变快了
7359a08e891641f745d052b4a9b25b95f5efe592 (origin/master, master) 可以手打神符
078793e10c5b54b0f6fbf8dfc10cd70a9e7da0c9 上场训练 补充了分区赛一些功能
b44832f8a6179a209fa775239fba67f0f4ca3ede 整套代码可以使用
efaa533152de25f9c848bac94ff08e5dc243aa17 新车可以读裁判系统数据进行热量和功率限制
...
```

撤消操作

有时候提交完代码之后，你会发现忘记修改其中的某个文件，或者写错了提交信息，这个时候使用 `--amend` 参数来尝试重新提交；

```
$ git commit -m 'initial commit'
$ git add forgotten_file
$ git commit --amend
```

如果没有修改任何文件，那么本次提交只是修改了上次提交的信息：

```
$ git commit -m "Merge 2 commits" --amend
[dev_rune 5991a8a] Merge 2 commits
 Date: Tue Sep 10 16:45:03 2019 +0800
```

取消暂存的文件

使用 `git reset HEAD <file>...` 命令来取消暂存：

```
$ git reset HEAD CONTRIBUTING.md
Unstaged changes after reset:
M       CONTRIBUTING.md
```

撤销对文件的修改（危险）

使用 `git checkout -- <file>...` 命令来撤消修改：

```
$ git checkout -- CONTRIBUTING.md
```

`git checkout -- [file]` 是一个危险的命令。 你对那个文件做的任何修改都会消失。 除非你确实清楚不想要那个文件了，否则不要使用这个命令。

远程仓库的使用

a) 添加远程仓库

使用命令 `git remote add <shortname> [url]` 来添加远程仓库

```
$ git remote add origin https://gitee.com/Anthracene/mnist_font_recognition.git
```

从此你可以用字符串 origin 来代替整个 URL

b) 抓取与拉取

使用命令 `git fetch [remote-name]` 来拉取本地没有的数据，执行后你将拥有远程仓库中所有分支的引用，可以随时合并或查看；
`git fetch` 命令会抓取数据到本地仓库，但是不会自动合并或修改你当前的工作，你必须手动将其合并；运行 `git pull` 则会从最初克隆的服务器上抓取数据并 **自动尝试合并** 到当前所在的分支。

c) 推送到远程仓库

使用命令 `git push [remote-name] [branch-name]` 将本地仓库的某个分支推送到远程仓库；

打标签

a) 列出标签

使用命令 `git tag` 列出所有标签；使用参数 `-l [keyword]` 来查找特定标签

```
$ git tag
V1.2.0
v1.0.0
...
$ git tag -l 'v1.8.5*'
v1.8.5
v1.8.5-rc0
v1.8.5-rc1
```

b) 打标签

使用参数 `-a` 来创建标签，使用参数 `-m [brief]` 来加上附注：

```
$ git tag -a v2.0.0 -m "重构完成"
```

c) 给过去的提交打标签

同样使用参数 `-a` 来创建标签，使用参数 `-m [brief]` 来加上附注，在最后加上指定提交的校验和（或部分校验和）

```
$ git log --pretty=oneline
1f45cf680c93b6ebcecf499b6d0be7721fd34a72 底盘和主控代码合并
1de8b43ef52827c295a8595e985be81948430215 修复了近距离打基地模式热量累计过快的bug
2947bec99610fc3e12a26e7e1558dd1e52ae6262 上场代码 打基地模式待测试
7aa9bdc14d60dfd3ddf03c3ea4c4995de74fd075 修改了开电容的时候的限幅
$ git tag -a v2.0.0 -m "重构完成" 1f45cf
```

5. 进阶使用

未完待续

详细资料请查阅 https://git-scm.com/book/zh/v2