1. **Git入门**
2. **Git 基础**

版本控制是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，可以对任何类型的文件进行版本控制。每一次的提取操作，实际上都是一次对代码仓库的完整备份。在 Git 中的绝大多数操作都只需要访问本地文件和资源，不用连网。在保存到 Git 之前，所有数据都要进行内容的校验和（checksum）计算，并将此结果作为数据的唯一标识和索引。Git 使用 SHA-1 算法计算数据的校验和，通过对文件的内容或目录的结构计算出一个 SHA-1 哈希值，作为指纹字符串，由 40 个十六进制字符组成。实际上，所有保存在 Git 数据库中的东西都是用此哈希值来作索引的，而不是靠文件名。

对于任何一个文件，在 Git 内都只有三种状态：已提交（committed），已修改（modified）和已暂存（staged）。已提交表示该文件已经被安全地保存在本地数据库中了；已修改表示修改了某个文件，但还没有提交保存；已暂存表示把已修改的文件放在下次提交时要保存的清单中。基本的 Git 工作流程如下：

1. 在工作目录中修改某些文件。
2. 对修改后的文件进行快照，然后保存到暂存区域。
3. 提交更新，将保存在暂存区域的文件快照永久转储到 Git 目录中。
4. **配置下Git 工作环境**

Git 提供了一个叫做 git config 的工具专门用来配置或读取相应的工作环境变量。

1. /etc/gitconfig 文件：系统中对所有用户都普遍适用的配置。若使用 git config 时用 --system 选项，读写的就是这个文件。
2. ~/.gitconfig 文件：用户目录下的配置文件只适用于该用户。若使用 git config 时用 --global 选项，读写的就是这个文件。
3. 当前项目的 git 目录中的配置文件（也就是工作目录中的 .git/config 文件）：这里的配置仅仅针对当前项目有效。每一个级别的配置都会覆盖上层的相同配置，所以 .git/config 里的配置会覆盖 /etc/gitconfig 中的同名变量。
4. **配置信息**

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

$ git config --global user.email [johndoe@example.com](mailto:johndoe@example.com)

$ git config --list

user.name= John Doe

user.email=[johndoe@example.com](mailto:johndoe@example.com)

有时候会看到重复的变量名，那就说明它们来自不同的配置文件（比如 /etc/gitconfig 和 ~/.gitconfig），不过最终 Git 实际采用的是最后一个。也可以直接查阅某个环境变量的设定，只要把特定的名字跟在后面即可

$ git config user.name

John Doe

1. **使用帮助**

* $ git help 命令名称
* $ git 命令名称 --help
* $ man git-命令名称

1. **使用 Git**
2. **取得 Git 项目仓库**

有两种取得 Git 项目仓库的方法，第一种是在现存的目录下，通过导入所有文件来创建新的 Git 仓库。第二种是从已有的 Git 仓库克隆出一个新的镜像仓库来。

* 在工作目录中初始化新仓库

1. 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

初始化后，在当前目录下会出现一个名为 .git 的目录，所有 Git 需要的数据和资源都存放在这个目录中。不过目前，仅仅是按照既有的结构框架初始化好了里边所有的文件和目录，但我们还没有开始跟踪管理项目中的任何一个文件。

1. 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'

* 从现有仓库克隆：Git 收取的是项目历史的所有数据（每一个文件的每一个版本），服务器上有的数据克隆之后本地也都有了。

克隆仓库的命令格式为 git clone [url]

$ git clone git://github.com/schacon/grit.git

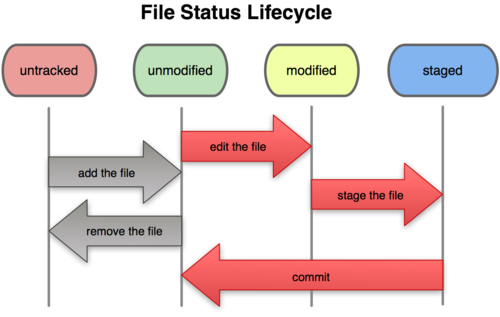
这会在当前目录下创建一个名为grit的目录，其中包含一个 .git 的目录，用于保存下载下来的所有版本记录，然后从中取出最新版本的文件拷贝。如果希望在克隆的时候，自己定义要新建的项目目录名称，可以在上面的命令末尾指定新的名字：

$ git clone git://github.com/schacon/grit.git **mygrit**

**Git 支持许多数据传输协议。之前的例子使用的是 git:// 协议，不过你也可以用 http(s):// 或者 user@server:/path.git 表示的 SSH 传输协议。**

1. **记录每次更新到仓库**

**工作目录下**面的所有文件都不外乎这两种状态：已跟踪或未跟踪。已跟踪的文件是指本来就被纳入版本控制管理的文件，在上次快照中有它们的记录，工作一段时间后，它们的状态可能是未更新，已修改或者已放入暂存区。而所有其他文件都属于未跟踪文件。它们既没有上次更新时的快照，也不在当前的暂存区域。初次克隆某个仓库时，工作目录中的所有文件都属于已跟踪文件，且状态为未修改。



在编辑过某些文件之后，Git 将这些文件标为已修改。我们逐步把这些修改过的文件放到暂存区域，直到最后一次性提交所有这些暂存起来的文件，如此重复。

* **检查当前文件状态**

$ git status

创建一个新文件后，保存退出后运$ git status 会看到该文件出现在未跟踪文件列表中。

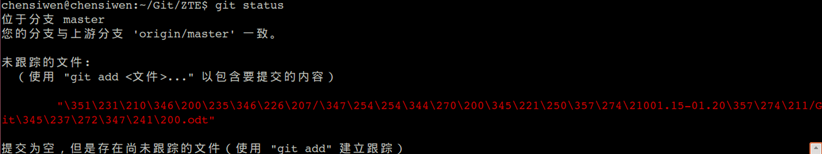
* 跟踪新文件：命令 git add 开始跟踪一个新文件。

$ git add 文件名

此时再运行 git status 命令，会看到文件已被跟踪，并处于暂存状态，即在 “Changes to be committed” 这行下面。在 git add 后面可以指明要跟踪的文件或目录路径。如果是目录的话，就说明要递归跟踪该目录下的所有文件。

**git status 中文文件名编码问题解决（Windows 、Linux）**

在默认设置下，中文文件名在工作区状态输出，查看历史更改概要，以及在补丁文件中，文件名的中文不能正确地显示，而是显示为八进制的字符编码，



通过将Git配置变量 core.quotepath 设置为false，就可以解决中文文件名称在这些Git命令输出中的显示问题



* **暂存已修改文件**

要暂存这次更新，需要运行 git add 命令（这是个多功能命令，根据目标文件的状态不同，此命令的效果也不同：可以用它开始跟踪新文件，或者把已跟踪的文件放到暂存区，还能用于合并时把有冲突的文件标记为已解决状态等。

运行了 git add 之后又作了修订的文件，需要重新运行 git add 把最新版本重新暂存起来

* **忽略某些文件**

有些文件无需纳入 Git 的管理，也不希望它们总出现在未跟踪文件列表。通常都是些自动生成的文件，比如日志文件，或者编译过程中创建的临时文件等。可以创建一个名为 .gitignore 的文件，列出要忽略的文件模式。

文件 .gitignore 的格式规范如下：

1. 所有空行或者以注释符号 ＃ 开头的行都会被 Git 忽略。
2. 可以使用标准的 glob 模式匹配。
3. 匹配模式最后跟反斜杠（/）说明要忽略的是目录。
4. 要忽略指定模式以外的文件或目录，可以在模式前加上惊叹号（!）取反。

所谓的 glob 模式是指 shell 所使用的简化了的正则表达式。

* 星号（\*）匹配零个或多个任意字符；
* [abc] 匹配任何一个列在方括号中的字符（要么匹配一个 a，要么匹配一个 b，要么匹配一个 c）；
* 问号（?）只匹配一个任意字符；
* 如果在方括号中使用短划线分隔两个字符，表示所有在这两个字符范围内的都可以匹配（比如 [0-9] 表示匹配所有 0 到 9 的数字）。
* **查看已暂存和未暂存的更新**

查看具体修改了什么地方，可以用 git diff 命令。

当前做的哪些更新还没有暂存？有哪些更新已经暂存起来准备好了下次提交？

git diff 会使用文件补丁的格式显示具体添加和删除的行。

* 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：比较的是工作目录中当前文件和暂存区域快照之间的差异，也就是修改之后还没有暂存起来的变化内容。
* 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。

* **提交更新**

1. 每次准备提交前，先用 git status 看下，是不是都已暂存起来了，然后再运行提交命令 git commit
2. 可以用 -m 参数后跟提交说明的方式

git commit -m "Story 182: Fix benchmarks for speed"

1. 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

* 移除文件

1. 用 git rm 命令：从 Git 中移除某个文件，就必须要从已跟踪文件清单中移除（确切地说，是从暂存区域移除），然后提交，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。
2. 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 “Changes not staged for commit” 部分（也就是未暂存清单）看到 🡪然后再运行 git rm filename记录此次移除文件的操作 🡪最后提交的时候，该文件就不再纳入版本管理了
3. 把文件从 Git 仓库中删除（亦即从暂存区域移除），但仍然希望保留在当前工作目录中。换句话说，仅是从跟踪清单中删除，要移除跟踪但不删除文件。

git rm --cached 文件名

* 重命名文件

$ git mv file-a file-b

* **查看提交历史**

**$ git log**

**$ git log -p -2：-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。在做代码审查，或者要快速浏览其他协作者提交的更新都作了哪些改动时，就可以用这个选项**

* **撤消操作**

$ git commit --amend重新提交，将使用当前的暂存区域快照提交

* 取消已经暂存的文件

git reset HEAD <file>...

* 取消对文件的修改

$ git checkout -- <file>

取消修改，回到之前的状态（也就是修改之前的版本）

* **远程仓库的使用**

**$ git remote：查看当前配置有哪些远程仓库**

**$ git remote -v：显示对应的克隆地址。如果有多个远程仓库，此命令将全部列出**

**$ git remote add [shortname] [url]：添加一个新的远程仓库，可以指定一个简单的名字，以便将来引用**

**$ git fetch [remote-name]：从远程仓库抓取数据到本地**

**$ git push [remote-name] [branch-name]** **将本地仓库中的数据推送到远程仓库。如果在你推数据前，已经有其他人推送了若干更新，那你的推送操作就会被驳回。你必须先把他们的更新抓取到本地，合并到自己的项目中，然后才可以再次推送。**

**$ git remote show origin查看某个远程仓库的详细信息**