# 1 Git

## 1.1 Git Install

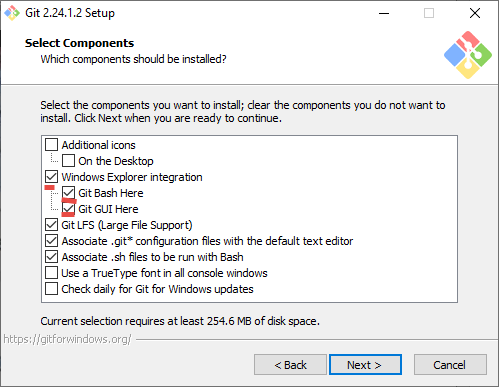
### 1.1.1 Install

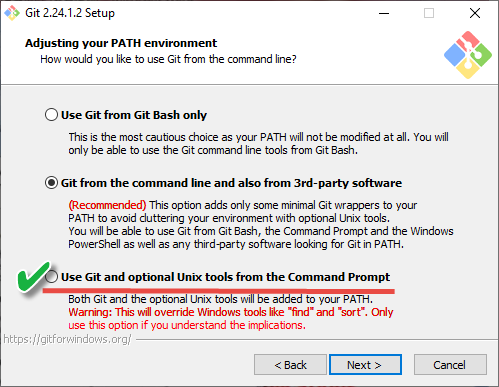
URL: <https://git-scm.com>

Version: Git-2.24.1.2-64-bit

**Note**:

1. Select components when install Windows Explorer integration -> Git Bash Here & Git GUI Here
2. 在进行下一步安装时一定要将Git的可用命令设置到环境属性之中（如果不选只能手动进行配置文件的操作）。





## 1.2 Deploy Configuration

**Concrete Content**：

1、进行全局信息的设置；

2、进行仓库的配置；

3、进行项目的代码控制；

4、 修改的撤销与删除。

### 1.2.1 Set Global Info

如果想要进行软件的开发，那么一定需要存在有软件的开发者人员信息。所以在使用Git之前，需要配置一些公共的环境属性。

1、设置开发者的用户名称

$ git config –global user.name Michael

2、设置开发者的邮箱信息

$ git config –global user.email Michael.Song@huameisoft.cn

3、获取Git环境下全局信息

$ git config –list (git config -l)

### 1.2.2 Create Repository

如果使用SVN需要初始化一个仓库目录，之后将此目录作为项目的提交路径；而在使用Maven时也需要有一个读取仓库。Therefore，在任何的软件版本控制工具中，仓库非常重要。

1、假如说在C:\gitpro的目录，这个目录要将其设置为仓库目录；

md C:\gitpro

2、如果想要将此目录变为仓库目录，则可以进入到此目录之中，进行配置；

cd gitpro

3、初始化仓库，对于仓库的初始化方式有两种

A、第一种：创建一个包含有配置信息目录的仓库文件夹；

$ git init

|  |
| --- |
| **Result**: Initialized empty Git repository in C:/gitpro/.git/ |

**Note**：会在此目录项创建一个名为”.git”的隐藏文件夹，其中包含有全部Git相关的版本信息。

B、第二种：直接将当前目录设置为Git仓库（自己搭建git远程服务时使用）；

$ git init –bare

|  |
| --- |
| **Result**: Initialized empty Git repository in C:/gitpro/ |

**Note**: 在此目录中不再生成”.git”的隐藏文件，meanwhile C:/gitpro 将保存所有的配置信息。

### 1.2.3 Basic Operation of Repository

在Git 中，可以监控的范围就是仓库信息（C:/gitpro/.git 本身就是一个仓库）。

**Operation**:

1、在C:\gitpro\下创建Hello.java文件（编码使用’ UTF-8’）。

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  System.out.println("Hello Git!");  }  } |

2、观察当前仓库状态

$ git status

|  |
| --- |
| **Result**:  On branch master  No commits yet  Untracked files:  (use "git add <file>..." to include in what will be committed)  Hello.java  nothing added to commit but untracked files present (use "git add" to track) |

**Note**：如果此时Git发现仓库中增加了新的文件或者修改了文件，则Git会提示用户文件的状态。

3、将Hello.java提交给Git的文件管理系统。

1）将文件添加到Git暂存区中。

$ git add Hello.java (FileName)

2）随后再次查询当前仓库的状态。

$ git status

|  |
| --- |
| **Result**:  On branch master  No commits yet  Changes to be committed:  (use "git rm --cached <file>..." to unstage)  new file: Hello.java |

**Note**: 此时Git知道了当前目录中有一个需要使用的新文件：Hello.java。

3）将文件提交到版本库中。

$ git commit –m “提交代码的注释信息”

$ git commit –m “Create New File Hello.java”

|  |
| --- |
| **Result**:  [master (root-commit) d3ad493] Create New File Hello.java  1 file changed, 5 insertions(+)  create mode 100644 Hello.java |

**Note**: 此时才表示将新的文件提交到了Git中进行管理（这个代码能够被项目真正使用）

**However**:

如果同时创建了好几个新文件，使用了”git status” 之后，会立刻告诉用户创建了N个新文件；

|  |
| --- |
| **Result**:  On branch master  Untracked files:  (use "git add <file>..." to include in what will be committed)  Demo.java.java  Emp.java.java  nothing added to commit but untracked files present (use "git add" to track) |

将多个文件添加到版本库中，传统做法：1> 使用”git add FILENSME”添加到暂存区中，2> 使用”git commit FILENAME” 提交到版本库中。

批量添加新文件到暂存库中：

$ git add .

此时，所有在仓库目录中创建的新文件均将自动保存到仓库的暂存区中。

|  |
| --- |
| **Result**:  On branch master  Changes to be committed:  (use "git restore --staged <file>..." to unstage)  new file: Demo.java.java  new file: Emp.java.java |

进行版本库的提交：

$ git commit –m “Add Two Files”

**Note**: 最终所有代码都要提交到版本库中才可以运行，那么完全没有必要先添加到暂存库，可以直接添加。

自动增加并提交到版本库中：

1) Edit Emp.java:

|  |
| --- |
| public class Emp implements java.io.Serializable  {  private Integer empNo;  } |

2) Inquire current status:

|  |
| --- |
| **Result**:  On branch master  Changes not staged for commit:  (use "git add <file>..." to update what will be committed)  (use "git restore <file>..." to discard changes in working directory)  modified: Emp.java.java  no changes added to commit (use "git add" and/or "git commit -a") |

3) Add files to repository **Directly**

$ git commit **–a –m** “Change Emp.java File”

|  |
| --- |
| **Result**:  [master 5b67db2] Change Emp.java File  1 file changed, 3 insertions(+), 4 deletions(-) |

**Note**: 不需要区分暂存库和版本库，直接将代码发布

### 1.2.4 Modify the repository file

针对于仓库中的文件进行修改的话，在查询状态时会进行及时的体现。

1）Update Hello.java

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  System.out.println("Hello Git!");  System.out.println("Hello World!");  System.out.println("Hello ~");  }  } |

如果想知道此时的文件与发布的代码那块发生了改变，则可以使用一下命令。

比较文件：

$ git diff FILE

$ git diff Hello.java

|  |
| --- |
| **Result**:  diff --git a/Hello.java b/Hello.java  index fa0e9a3..a7d0326 100644  --- a/Hello.java  +++ b/Hello.java  @@ -1,5 +1,7 @@  public class Hello {  public static void main(String args[]) {  System.out.println("Hello Git!");  - System.out.println("Hello ");  + System.out.println("Hello World!");  + System.out.println("Hello ~");  }  }  \ No newline at end of file |

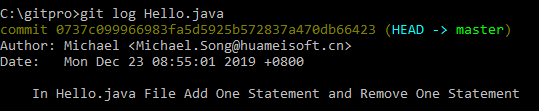
**Note**：如果出现的修改文件比较多，那么会采用换行的方式进行显示，如果最后不需要了使用”Q”退出。

**Note**：命令行的方式下，如果使用了”+”表示代码的增加，”-”表示代码的删除，此工具可以很清楚的帮助用户进行代码修改前后的区分。但修改完后的代码最终依然还是需要将其进行提交。

|  |
| --- |
| $ git commit -a -m "In Hello.java File Add One Statement and Remove One Statement" |

到此为止，已经出现了多次的提交，现在要想知道提交的历史，则使用一下命令查看：

$ git log FileName

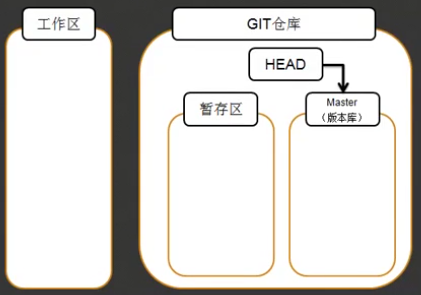


**Note**：每次提交时，每次都会自动生成一个Commit ID（在日后进行版本恢复时使用）

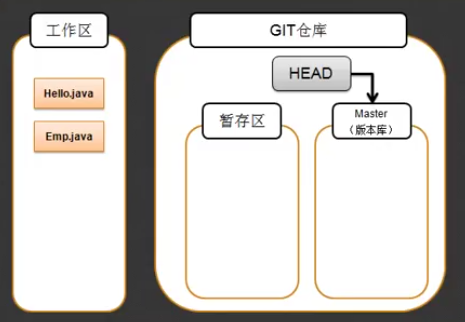
## 1.3 Work Directory、Staging Area、Repository

在整个Git中对于文件的操作一共提供有三个区域：

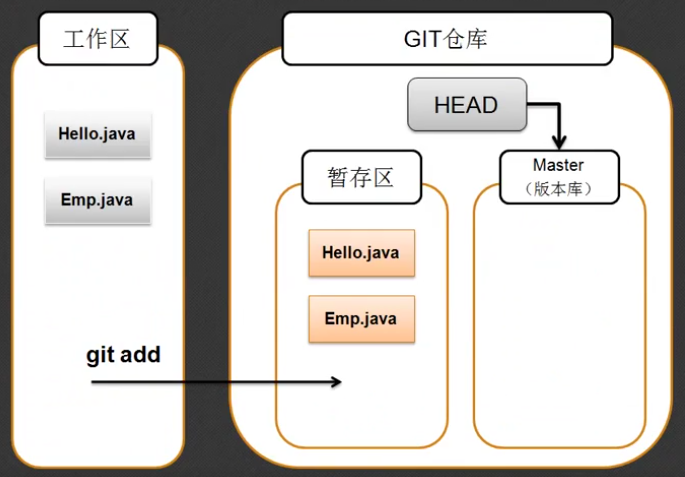
1. 工作区（用户所编写的代码文件夹）：所有的文件操作都以工作区为主；
2. GIT仓库：
   1. 暂存库：只是将工作区中的未保存文件保存到暂存区中，此部分由Git维护；
   2. 版本库：是真正进行项目发布的代码。



1、首先，用户要在工作区中编写所需要的程序文件，但是此时的文件不能够真正的保存在GIT仓库中，并且在这个区域中所进行的文件操作都不会影响最终的软件版本发布；相当于一个临时区域（工作环境）。

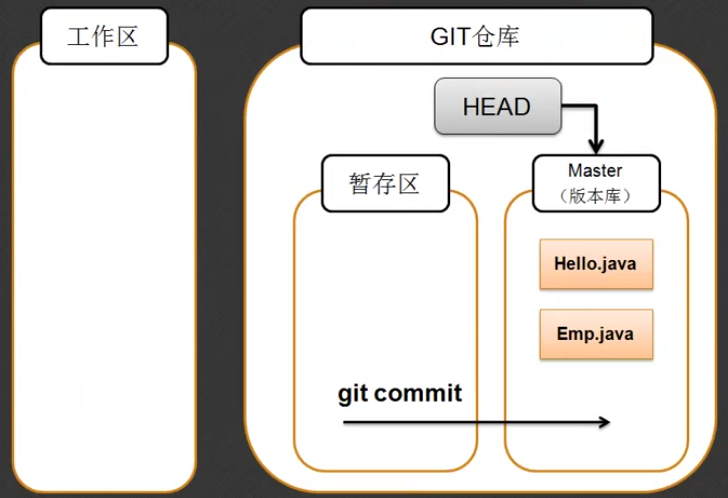


2、如果在需要将临时区域中的内容保存到最终发布的软件版本中，需要将其首先增加到暂存区（此时的代码时可以撤销的，并没有发布到版本库中），使用”$ git add .”进行增加。



使用”$ git add .”之后，工作区中的新文件的状态进行了删除，而后将所有的新文件保存到暂存区中。

3、保存在暂存区中的内容只能说是暂时需要的，那么也可能从暂存区之中的内容删除掉，而后要想真正能够发布项目，必须将其设置在Master分支上（版本库）。只有进行提交之后才可以将暂存区中的内容保存在版本库中，供其他人使用。



**Note**：由于一个项目可能会被发布多次，所以在Git中进行了保存之后，都会自动生成一个提交的版本编号（Commit ID），用于为开发者提供版本回退的版本机制。

**Operation**：

1）修改Hello.java文件；

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  System.out.println("Hello~");  System.out.println("Hello Git!");  System.out.println("Hello World!");  System.out.println("Hello Test！"); // Add a new statement.  }  } |

此时，文件内容在工作区中修改，这个文件不能被真正的项目所使用，现在整个工作区的状态发生了变化。

2）当使用$ git status 操作时，会进行提示，提示工作区有变化；

|  |
| --- |
| **Result**：  On branch master  Changes not staged for commit:  (use "git add <file>..." to update what will be committed)  (use "git restore <file>..." to discard changes **in working directory**)  modified: Hello.java  no changes added to commit (use "git add" and/or "git commit -a") |

3）需要进行暂存区的保存（$ git add .），保存完成之后继续观察状态；

|  |
| --- |
| **Result**：  On branch master  Changes to be committed:  (use "git restore --staged <file>..." to **unstage**)  modified: Hello.java |

实际上，保存到暂存区中的内容，只是给了用户一个暂时不改代码的空间，如果用户现在发现修改错误了，则可以进行恢复。

4）如果现在继续修改Hello.java文件；

|  |
| --- |
| public class Hello {  public static void main(String args[]) {  System.out.println("Hello~");  System.out.println("Hello Git!");  System.out.println("Hello World!");  System.out.println("Hello Test！");  System.out.println(new java.util.Date()); // Add a new statement.  }  } |

此时的修改是在工作区中，与暂存区没有任何关系，因为没有使用（$ git add .）命令。当查看如下状态时，可以看到如下信息：

|  |
| --- |
| **Result**：  On branch master  Changes to be committed:  (use "git restore --staged <file>..." to **unstage**)  modified: Hello.java  Changes not staged for commit:  (use "git add <file>..." to update what will be committed)  (use "git restore <file>..." to discard changes **in working directory**)  modified: Hello.java |

此时，暂存区的文件没有提交到master上，另外工作区的文件也进行了修改，两个区域的文件是不同版本的两个文件。

5）如果此时提交项目到版本库中（Master分支），工作区的修改不应该被提交；

|  |
| --- |
| $ git commit -m "Commit Work Directory File"  **Result**：  [master 8cb70a7] Commit Work Directory File  1 file changed, 1 insertion(+) |

此时，只将暂存区中的代码提交到了Master分支上，工作区的文件并未提交。提交的只是暂存区中的内容，而工作区的内容并不会受到影响。

|  |
| --- |
| **Result**：  On branch master  Changes not staged for commit:  (use "git add <file>..." to update what will be committed)  (use "git restore <file>..." to discard changes in working directory)  modified: Hello.java  no changes added to commit (use "git add" and/or "git commit -a") |

**Summary**：

真正能使用的代码在Master分支上；

暂时要修改的代码可以保存在暂存区中；

工作区中的代码是没有用的（相当于编译器中的代码并没有发送到Git服务器中）。

## 1.4 Cross Version

Git中最强大的操作就在于可以进行版本的回退以及前进的状态下切换。在任何的开发之中，对于软件的代码都很难保证其不恢复到原始状态。因为新增加的程序可能会由于某些原因从而导致失败，此时必须要能够进行快速的版本回退。

Git进行版本穿越的核心概念在于每一个“Commit ID”。

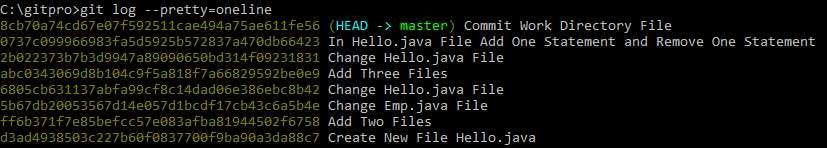
1）查看当前的日志信息：

$ git log FileName

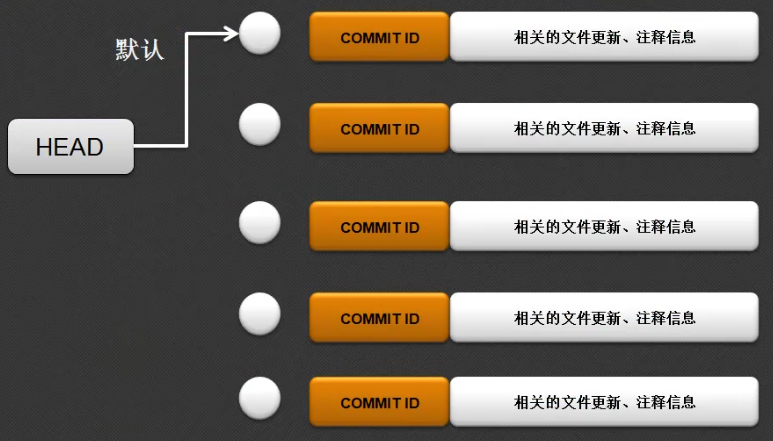
可以发现整个的日志记录采用的使栈的形式，最早的提交保存在最后。

简洁显示（只能以当前提交点的记录为主【即：只能查询到当前提交点之前的全部操作，修改HEAD之后的其他记录（更换提交点之后的操作）均查不到】）：

$ git log –pretty=oneline



实际上，在Master分支上存在有一个HEAD指针，这个HEAD指的是当前正在使用的版本。而HEAD指正在默认情况下，一定指向的是最新的提交点。

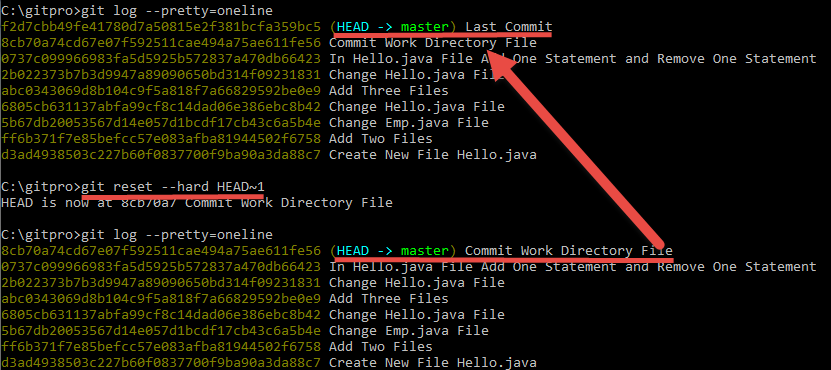


如果说现在要是改变了HEAD的指针，那么就相当于可以任意的修改软件的版本。



回退到上一个版本：

$ git reset –hard HEAD~1

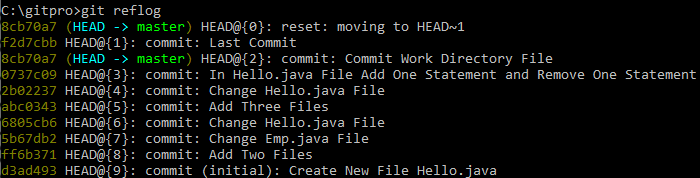


实际上，此时就实现了版本的回退操作，每当回退一次都回到了上一次的保存点（Commit ID）。如果回退过了，则需要rollback回去，这个时候就要知道提交点的信息。

但是依靠传统的日志查询已经不可行，必须依靠一个删除的提交点日志信息。

查看所有删除的提交点：

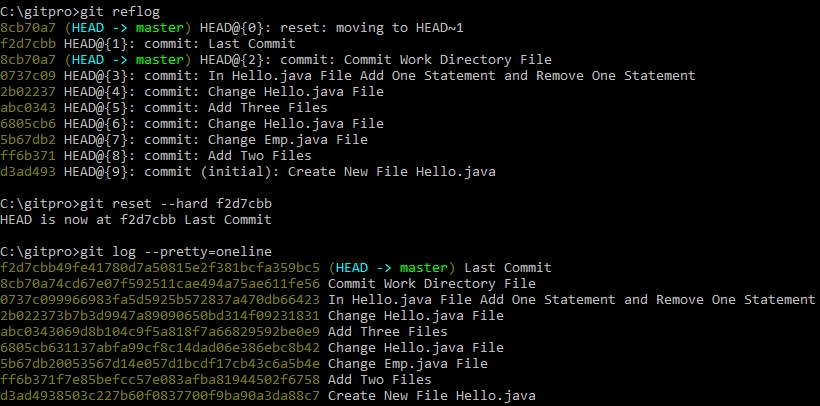
$ git reflog

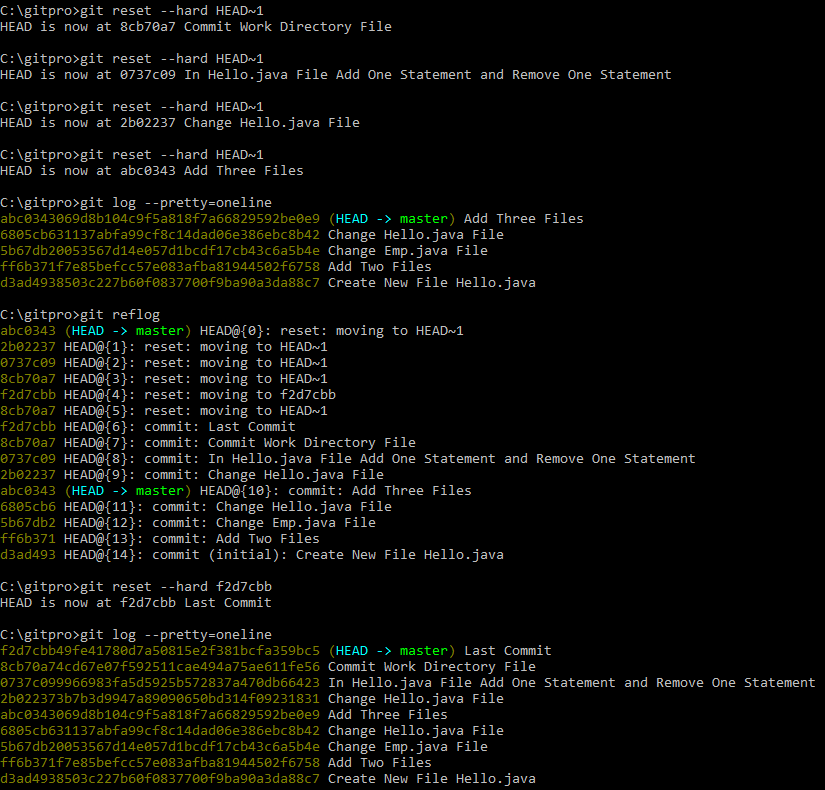


发现所有针对于版本的操作，$ git reflog 都可以记录好，那么就可以实现恢复了。

恢复到最后一次提交点（eg：f2d7cbb）：

$ git reset –hard f2d7cbb





**Note**：只要有正常的提交，所有的提交点就可以作为恢复点存在。

## 1.5 Undo modify

在任何的开发过程中，都很难保证不会发生“无用修改”的操作。而在Git中针对此类操作也有所支持，实际上就是指修改撤销的功能，可是需要考虑两种情况：

Case 1:

工作区中发生了改变，但是没有提交到暂存区（未 add & commit）；

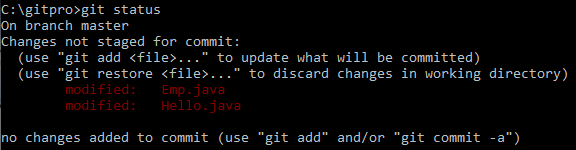
Case 2：

已经添加到了暂存区，但是没有提交到Master分支（已add Not commit）。

1、在工作区中进行开发（未add & commit）

这个时候如果要想恢复原本的代码，可以使用restore（checkout）操作。

首先使用”$ git status”查看当前状态



查看哪些文件发生了变化：

$ git checkout

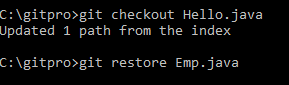


需要重新检出已有的代码（恢复文件）：

$ git restore FileName（新版本）

或者：

$ git checkout FileName（旧版本）



restore 恢复代码不能一次性恢复，只能手动一次性恢复一个文件。

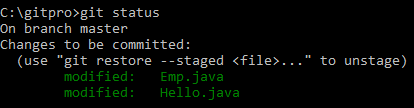
如果真的想一次性恢复，把要回退的代码全部删除，然后commit，之后回滚到上一个提交点即可。

2、工作区已修改，暂存区已保存，马上就要提交了，发现出错了。

首先修改Emp.java和Hello.java文件（eg：将文件内容全部删除）；

将文件保存到暂存区中（$ git add .）；

查询当前状态（$ git status）；



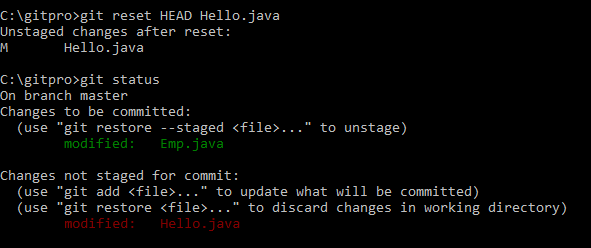
将保存在暂存区中的文件踢回到工作区中：

$ git reset HEAD Hello.java

将保存在暂存区中的Hello.java文件恢复到工作区中



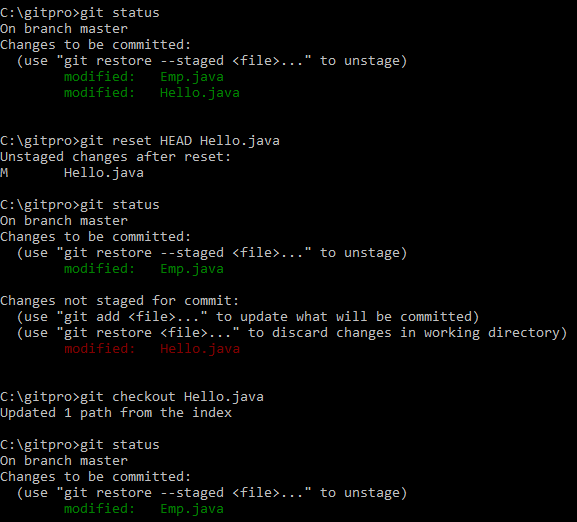
恢复完成之后，查看当前状态：



此时，已经修改了两个文件，Hello.java由暂存区被踢回到了工作区中（not staged for commit），Emp.java目前处于暂存区并没有提交到Master分支。

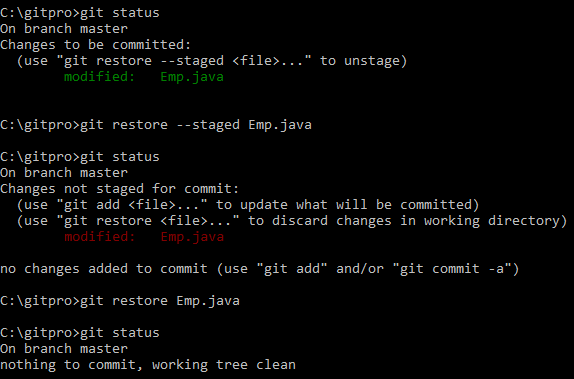
随后需要恢复已有的数据：

$ git checkout Hello.java（或者$ git restore –staged Hello.java）



**旧版本执行命令**

Hello.java已经恢复到原本状态。



**新版本执行命令**

同理，将Emp.java回退到工作区，再restore回上一个版本。

**Summary**:

工作区 → 暂存区：$ git add

工作区 → Master分支：$ git commit

暂存区 → 工作区：$ git restore –staged FileName（新版本）

$ git reset HEAD FileName（旧版本）

工作区（or 暂存区）恢复上一版本数据：$ git restore FileName（新版本）

$ git checkout FileName（旧版本）

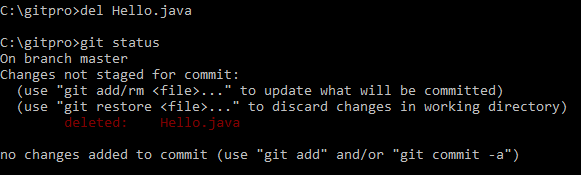
## 1.6 Delete File

一定在开发中发现文件误删的情况，那么现在如果想要进行文件的删除，实际上只需要删除掉工作区的内容即可。

删除文件：

del Hello.java （windows 命令）

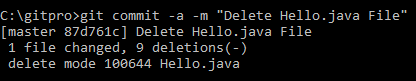
随后查询当前状态：



此时，已经检查到了Hello.java文件已经被删除，此时只是在工作区中被删除，但是在Master分支上依然存在有Hello.java文件，所以必须进行更新提交。

更新提交删除文件：

$ git commit –a –m “Delete Hello.java File”



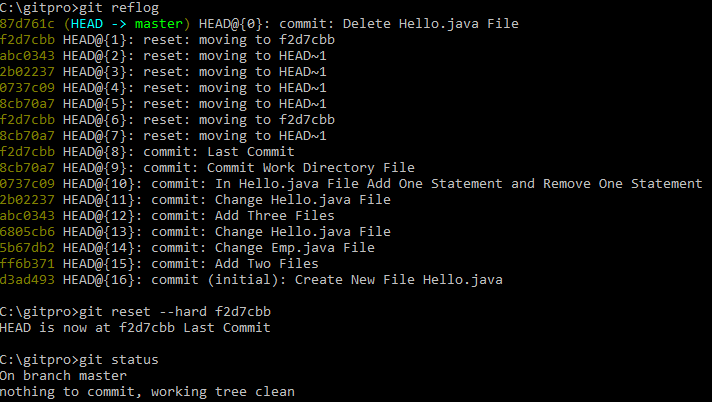
只要此时更新一提交，那么就表示Hello.java文件彻底消失了。

但是此时认为Hello.java文件删除错了，那么可以进行恢复（Git版本穿越）。

$ git reflog

恢复文件（恢复到之前的某一个版本）：

$ git reset –hard f2d7cbb



**以上的操作**指的是删除已经被更新提交的情况，那么还有一种case：

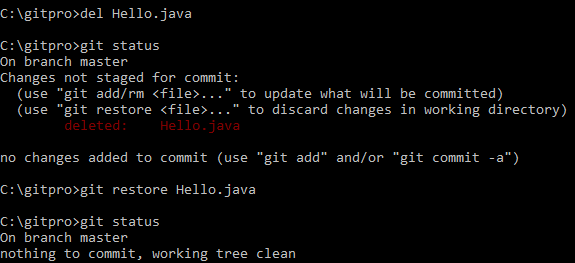
删除发生在工作区中；

这时由于没有提交，所以Master分支上依然存在有Hello.java文件，那么可以直接利用Hello.java文件进行恢复。

$ git restore Hello.java（新版本）

Or

$ git checkout Hello.java（旧版本）



**Summary**:

工作区中的删除可以利用restore（checkout） 重新检出；

Master分支上的删除就必须利用版本穿越的方式恢复。

# 2 GitHub

**Concrete Content**:

1、注册GitHub账号；

2、配置、合并、克隆远程仓库。

## 2.1 Description of GitHub

在之前的Git讲解中，Git并没有使用的严格的网络要求。但是个人编写完成的代码，如果最终要想给他人使用，那么需要将项目公布到网络上，所以在Git创建之初也考虑到了此类问题，专门提供了一个网站[www.github.com](http://www.github.com)，该网站就是一个Git公共服务器。

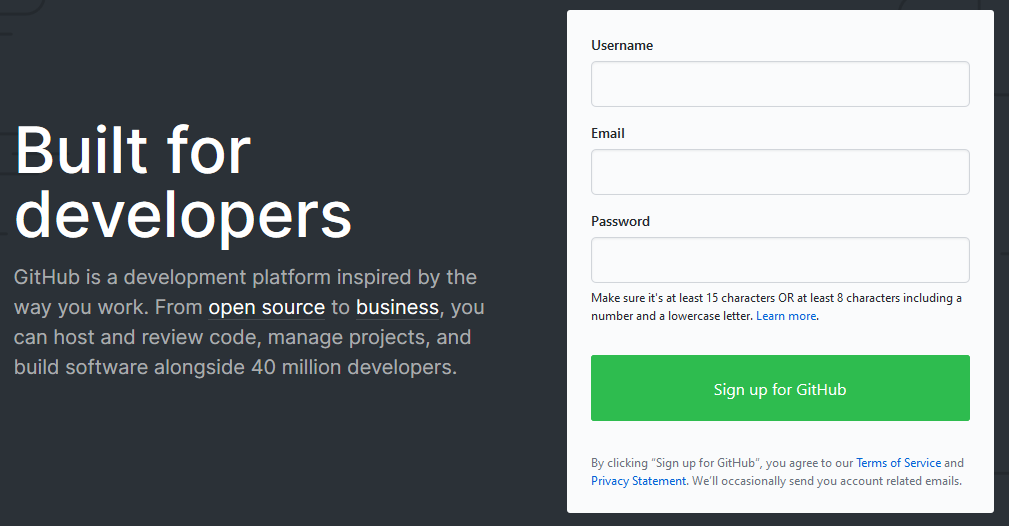
在这个网站上进行项目发布的时候需要注意一点，它可以发布两类的项目：

Type 1：

公共项目（免费）：可能需要将自己的代码交给其他人进行完善；

Type 2：

私有项目（收费）：作为一个公司的开发项目。



在页面中输入注册的邮箱和密码。

## 2.2 Configuring the remote repository

GitHub就是一个服务器，它可以直接保存各个客户端发送的数据。但是如果想要进行保存数据的操作，必须要使用SSH的通讯模式，需要的配置如下：

12018-01:00