

一、 Git 下载和安装：

- (1) 从 git 官网上 <https://git-scm.com/downloads> 下载和安装对于系统 (windows、Linux 和 Mac) 的 git，默认安装即可，但由于官网上的 git 的镜像源在国外，下载速度非常慢，甚至可能下载失败，因此推荐第二种方式；
- (2) <https://npm.taobao.org/mirrors/git-for-windows/>，可通过该镜像源（该镜像源在国内，下载速度非常快）下载安装相应版本的 git；

git 下载安装完成后，接下来可以使用 git 指令将本地仓库代码上传到 GitHub 或 GitLab 远程仓库

二、 git 上传代码或文件至 GitHub 远程仓库

- (1) 在 GitHub 或者 GitLab 上手动创建新的 repository，本文基于 GitHub 操作，创建如下图所示：

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner *



wujunming1

Repository name *

hello-introduction



Great repository names are short and memorable. Need inspiration? How about [stunning-memory?](#)

Description (optional)



Public

Anyone on the internet can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

- (2) 创建好之后，然后复制一下远程仓库的 https 地址，如下所示：

wujunming1 / hello-introduction

Unwatch 1 Star 0

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

Quick setup — if you've done this kind of thing before

Set up in Desktop or [HTTPS](#) [SSH](#) <https://github.com/wujunming1/hello-introduction.git>

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

- (3) 在你的本地创建一个空的目录，用来与 GitHub 的远程仓库相关联（我这里创建时 test 文件夹），使用 git 终端切换到 test 目录下，通过 git clone <https://github.com/wujunming1/hello-introduction.git> 命令将 GitHub 远程仓库上的代码和文件克隆下来，此时本地 test 目录下就出现了一个 hello-introduction 文件夹（其实就是

GitHub 上创建的仓库名)，这个文件夹已经不是简单普通的文件夹，它现在完全是由 git 管理，包括文件的增加、修改和删除都是由 git 进行记录和跟踪，这样一个文件夹也称为 git 本地仓库。而且，通过这一指令之后本地仓库与 GitHub 远程仓库上相关联了，也就是我们通过在本地使用 git 指令对远程仓库 GitHub 上的文件更新和删除了。

```
yizhou@LAPTOP-TSRBA3R MINGW32 /c/test
$ git clone https://github.com/wujunming1/hello-introduction.git
Cloning into 'hello-introduction'...
warning: You appear to have cloned an empty repository.
```

我们接着可以尝试使用 git branch -vv 查看一下本地仓库的 master 分支与远程仓库 GitHub 上的 master 是否已经关联。可以看到左边是本地仓库的 master 分支，右边的是远程仓库 master 分支，二者已经进行了关联。换句话说，本地仓库已经和远程仓库已经建立好联系了。**注：还可通过以下命令行的方式把本地普通文件夹变成 git 管理的本地仓库并并远程仓库相互关联（GitHub 远程创建了空的“123”仓库）。**

```
echo "# 123" >> README.md
```

```
git init
```

```
git add README.md
```

```
git commit -m "first commit"
```

```
git remote add origin https://github.com/wujunming1/123.git
```

```
git push -u origin master
```

```
yizhou@LAPTOP-TSRBA3R MINGW32 /c/123-test
$ echo "# 123" >> README.md
git commit -m "first commit"
git remote add origin https://github.com/wujunming1/123.git
git push -u origin master
yizhou@LAPTOP-TSRBA3R MINGW32 /c/123-test
$ git init
Initialized empty Git repository in C:/123-test/.git/

yizhou@LAPTOP-TSRBA3R MINGW32 /c/123-test (master)
$ git add README.md
warning: LF will be replaced by CRLF in README.md.
The file will have its original line endings in your working directory

yizhou@LAPTOP-TSRBA3R MINGW32 /c/123-test (master)
$ git commit -m "first commit"
[master (root-commit) da8b98b] first commit
1 file changed, 1 insertion(+)
create mode 100644 README.md

yizhou@LAPTOP-TSRBA3R MINGW32 /c/123-test (master)
$ git remote add origin https://github.com/wujunming1/123.git
```

```
yizhou@LAPTOP-TSRBA3R MINGW32 /c/123-test (master)
$ git remote add origin https://github.com/wujunming1/123.git

yizhou@LAPTOP-TSRBA3R MINGW32 /c/123-test (master)
$ git push -u origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 212 bytes | 106.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/wujunming1/123.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.

yizhou@LAPTOP-TSRBA3R MINGW32 /c/123-test (master)
$ git branch -vv
* master da8b98b [origin/master] first commit
```

注意：分支的概念，创建、删除、如何切换分支等在之后进行概述。

```
yizhou@LAPTOP-TSRBA3R MINGW32 /c/test/hello-introduction (master)
$ git branch -vv
* master f984ad9 [origin/master] new add
```

(4) 在本地仓库上，我们先假设创建一个 c.txt 文件，这时我们可以使用 git status 查看本地仓库的变化。可以看到 git 提示我们新创建一个文件，是否要使用 git add 指令提交文件。

```
yizhou@LAPTOP-TSRBA3R MINGW32 /c/test/hello-introduction (master)
$ touch c.txt

yizhou@LAPTOP-TSRBA3R MINGW32 /c/test/hello-introduction (master)
$ git status
On branch master

No commits yet

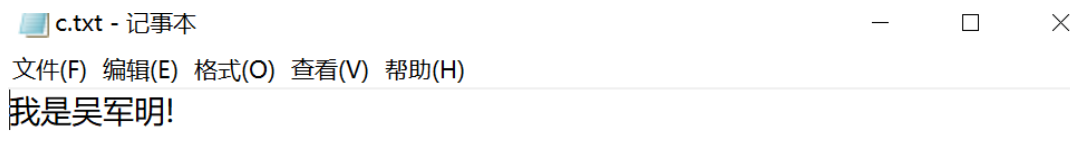
Untracked files:
  (use "git add <file>..." to include in what will be committed)
  c.txt

nothing added to commit but untracked files present (use "git add" to track)
```

此电脑 > Windows-SSD (C:) > test > hello-introduction

名称	修改日期	类型
.git	2020/8/8 15:08	文件夹
c.txt	2020/8/8 15:07	文本文档

我们接着再对 c.txt 进行修改操作，添加了“我是吴军明!”这段话，我们再用 git status 查看一下变化，可以看到 git 已经提示我们 c.txt 文件进行了修改，是否提交使用 git add 或 git commit 提交新的修改。



```
yizhou@LAPTOP-TSRBA3R MINGW32 /c/test/hello-introduction (master)
$ git status
On branch master
Your branch is up to date with 'origin/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:   c.txt

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

我们还可以使用 git diff c.txt 指令查看增加和删除了哪些内容。甚至可以使用 git checkout - c.txt 撤销你所做的修改，回到最初提交之前的版本。这些指令在之后会详细讲解。

```
yizhou@LAPTOP-TSRBA3R MINGW32 /c/test/hello-introduction (master)
$ git diff c.txt
diff --git a/c.txt b/c.txt
index e69de29..37da7b8 100644
--- a/c.txt
+++ b/c.txt
@@ -0,0 +1 @@
+我是吴军明!
\ No newline at end of file
```

(5) 接下来就是提交文件的流程了，我们使用 `git add c.txt` 指令（`git add .` 是提交新创建/修改的所有文件）提交新创建的文件 `c.txt`，`add` 指令是将本地工作区的文件内容提交到暂存区（缓存区）。若没报错，说明提交成功。

```
yizhou@LAPTOP-TSR5BA3R MINGW32 /c/test/hello-introduction (master)
$ git add .
```

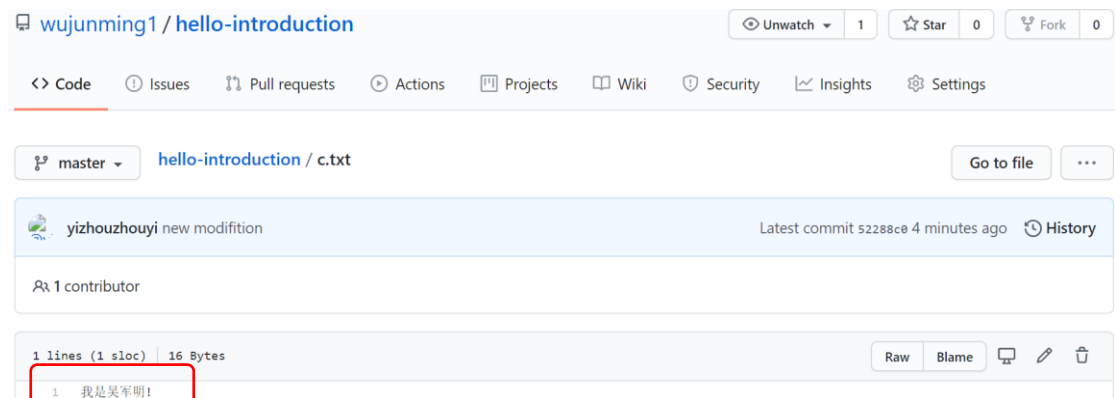
(6) 接着再使用 `git commit -m "文件提交说明"` 指令提交文件到本地仓库。

```
yizhou@LAPTOP-TSR5BA3R MINGW32 /c/test/hello-introduction (master)
$ git commit -m "new add"
[master (root-commit) f984ad9] new add
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 c.txt
```

(7) 最后关键的一步是，如何将本地仓库的 `master` 分支下的代码或文件（这里是 `c.txt`）推送到远程仓库 GitHub 上。使用 `git push -u origin master` 指令即可完成此次操作。很明显 `git push` 是推动的意思，`-u` 就是用户的意思，`origin` 有源头的意思，也就是远程仓库的意思，最后 `master` 就是分支名。每次使用该指令时需要输入 GitHub 的用户名和密码，至此我们把修改好的 `c.txt` 文件推到了 GitHub 远程仓库。

```
yizhou@LAPTOP-TSR5BA3R MINGW32 /c/test/hello-introduction (master)
$ git push -u origin master
fatal: HttpRequestException encountered.
  发送请求时出错。
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Writing objects: 100% (3/3), 252 bytes | 126.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/wujunming1/hello-introduction.git
   f984ad9..52288c0  master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

&&*&：可能你会有疑问，`git` 是如何进行版本管理、协同开发的呢，看上面一些指令操作不就是把文件从本地仓库推送到 GitHub 远程仓库上吗，怎么就能进行协同开发了？最开始的时候我们就说了，使用 `git clone` 指令把代码从远端仓库克隆到本地的時候，本地 `git` 仓库就已经和 GitHub 仓库关联了。现在假设有三个人协同开发一个代码，A 负责开发 A 功能，B 负责开发 B 功能，C 负责开发 C 功能。如果没有 `git` 管理的话，三个人修改后的代码只能通过 U 盘拷贝的方式合并，而有了 `git` 这一牛逼的东西之后，合并代码就变得异常方便和简单。



具体来说就是，A 修改完 A 功能的代码后直接可以把修改后的代码提交到远端仓库，这个时候 B、C 可能还在用以前的代码开发，这样就可能出现冲突。所以 B 和 C 在开发之前，一定要先使用 `git pull` 指令，把远端仓库的代码或文件给拉下来，确保是最新的代码，最后修改各自功能后提交代码即可。可以看出来，有了 `git` 指令，省去代码合并时磨嘴皮的功夫，一

个指令就能全部搞定!!!

```
yizhou@LAPTOP-TSRBA3R MINGW32 /c/test/hello-introduction (master)
$ git pull
Already up to date.
```

使用 git pull 指令后，提示已经是最新的代码。如果协同开发时，有人提交和修改了代码和文件，那么它会把修改的代码和文件从远端拉下来。

(8) 总结来说，git status、git add、git commit、git push 和 git pull 这些基础指令就足够使用了。

三、Git 分支概念、查看、创建、删除、切换分支

什么是 git 分支呢？可以理解为树的节点，每个节点有很多子节点。Git 仓库有一个最大的 master 分支，称为主分支。之后创建的分支都是子分支。为啥要有分支这一概念，一个 master 分支不就够了，为啥还要子分支？。其实你很容易想到，当项目很大时，需要大量的人协同开发，那么只有一个主分支的时候，开发效率很容易受到影响，开发模式还是属于串行的。而且只有一个分支时很容易出现代码合并冲突。有了子分支之后，大家可以在自己创建的子分支上参与开发自己负责的功能，最后每个人在各自分支上编写的代码统一在 master 主分支合并。这样一种开发模式是不是就变成并行的呢，互不影响，大大提升了开发的效率。其实分支并不是什么高级的东西，当项目很小的时候，一个 master 主分支就足够了。

接下来介绍一下 git 分支创建、删除和切换分支的一些指令。

(1) 查看分支：git branch -a 查看所有分支，包括本地和远程分支；git branch -r 查看远程分支；git branch -l 查看本地分支；

```
yizhou@LAPTOP-TSRBA3R MINGW32 /c/test/hello-introduction (master)
$ git branch -a
* master
remotes/origin/master
```

```
yizhou@LAPTOP-TSRBA3R MINGW32 /c/test/hello-introduction (master)
$ git branch -r
origin/master
```

```
yizhou@LAPTOP-TSRBA3R MINGW32 /c/test/hello-introduction (master)
$ git branch -l
* master
```

(2) 创建分支：git branch “分支名”，没报错，说明创建分支成功了！

```
yizhou@LAPTOP-TSRBA3R MINGW32 /c/test/hello-introduction (master)
$ git branch "wujunming"
```

(3) 切换分支：使用 git checkout wujunming 指令可以从 master 分支切换创建好的 wujunming 分支上；

```
yizhou@LAPTOP-TSRBA3R MINGW32 /c/test/hello-introduction (master)
$ git checkout wujunming
Switched to branch 'wujunming'

yizhou@LAPTOP-TSRBA3R MINGW32 /c/test/hello-introduction (wujunming)
```

(4) 删除分支：git branch -d wujunming 指令删除创建的分支；

```
yizhou@LAPTOP-TSRBA3R MINGW32 /c/test/hello-introduction (wujunming)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

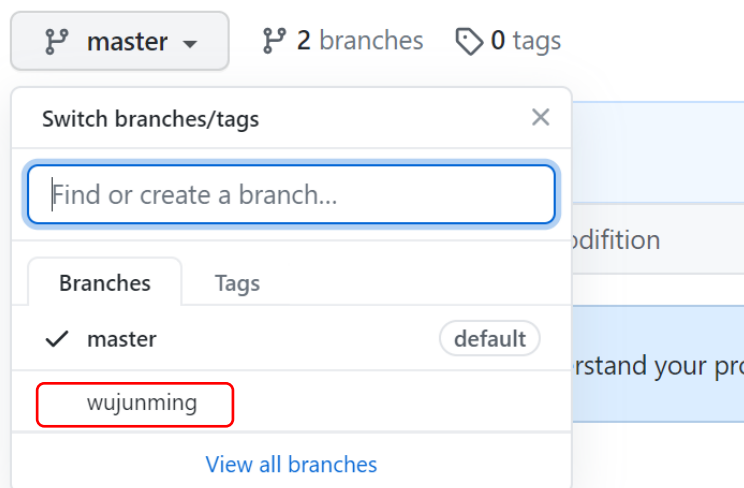
yizhou@LAPTOP-TSRBA3R MINGW32 /c/test/hello-introduction (master)
$ git branch -d wujunming
Deleted branch wujunming (was 52288c0).
```

创建的 wujunming 分支已经被删除了！

(5) 查看本地分支是否与远程分支相关联。git branch -vv，可以看到本地 master 分支已经和远程分支相互联系。

```
yizhou@LAPTOP-TSRBA3R MINGW32 /c/test/hello-introduction (master)
$ git branch -vv
master 52288c0 [origin/master] new modification
```

(6) 还有一条非常重要的指令就是如何创建本地分支并且与远程分支关联。假设远程仓库创建好了一个 wujunming 分支，那么我们如何在本地创建一个 wujunming 分支并且与远程 wujunming 关联起来呢？



远程仓库创建了新的分支，也就是远端代码或文件发生了变化，此时该怎么做？是不是应该先用 git pull 指令把最新的代码拉下来。

```
yizhou@LAPTOP-TSRBA3R MINGW32 /c/test/hello-introduction (master)
$ git pull
From https://github.com/wujunming1/hello-introduction
* [new branch]      wujunming -> origin/wujunming
Already up to date.
```

提示我们在远程仓库创建了一个新的分支 wujunming。我们再用 git branch -a 查看一下现在的分支情况。可以看到，远端仓库多了一个 wujunming 分支。

```
yizhou@LAPTOP-TSRBA3R MINGW32 /c/test/hello-introduction (master)
$ git branch -a
* master
remotes/origin/master
remotes/origin/wujunming
```

那么，最后我们如何在本地创建一个 wujunming 分支并且与远程 wujunming 分支关联呢？使用 git checkout -b wujunming origin/wujunming 指令，出现以下提示时说明关联成功了！我们再使用 git branch -a 查看所有分支情况。两个本地分支 master、wujunming，两个远程分支 origin/master，origin/wujunming。


```
yizhou@LAPTOP-TSRSBA3R MINGW32 /c/test/hello-introduction (master)
$ git checkout -b wujunming origin/wujunming
Switched to a new branch 'wujunming'
Branch 'wujunming' set up to track remote branch 'wujunming' from 'origin'

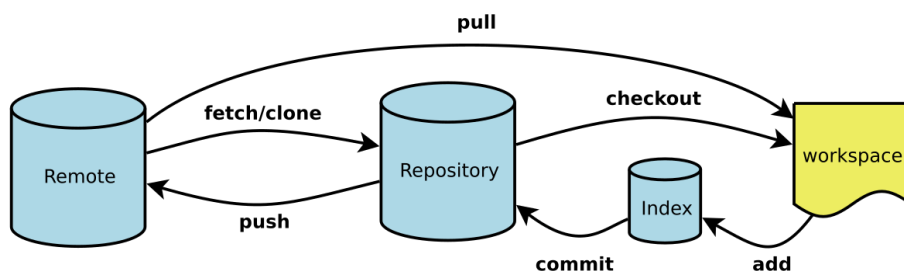
yizhou@LAPTOP-TSRSBA3R MINGW32 /c/test/hello-introduction (wujunming)
$ git branch -a
master
* wujunming
  remotes/origin/master
  remotes/origin/wujunming
```

最后再使用 `git branch -vv` 查看分支关联情况。本地 `wujunming` 分支已经和远程 `wujunming` 分支关联起来了！这样本地 `wujunming` 分支所做的修改可以同步到 `wujunming` 上了！至此分支相关的一些操作都介绍完毕了！

```
yizhou@LAPTOP-TSRSBA3R MINGW32 /c/test/hello-introduction (wujunming)
$ git branch -vv
master 52288c0 [origin/master] new modification
* wujunming 52288c0 [origin/wujunming] new modification
```

四、Git 其他一些指令（撤销修改和查看提交（git commit）日志）

基础知识：Git 本地有四个工作区域：工作目录（Working Directory）、暂存区（Stage/Index）、资源库（Repository 或 Git Directory）、git 仓库（Remote Directory）。文件在这四个区域之间的转换关系如下：



Workspace：工作区，就是你平时存放项目代码的地方；

Index / Stage：暂存区，用于临时存放你的改动，事实上它只是一个文件，保存即将提交到文件列表信息

Repository：仓库区（或版本库），就是安全存放数据的位置，这里面有你提交到所有版本的数据。其中 HEAD 指向最新放入仓库的版本

Remote：远程仓库，托管代码的服务器，可以简单的认为是你项目组中的一台电脑用于远程数据交换。

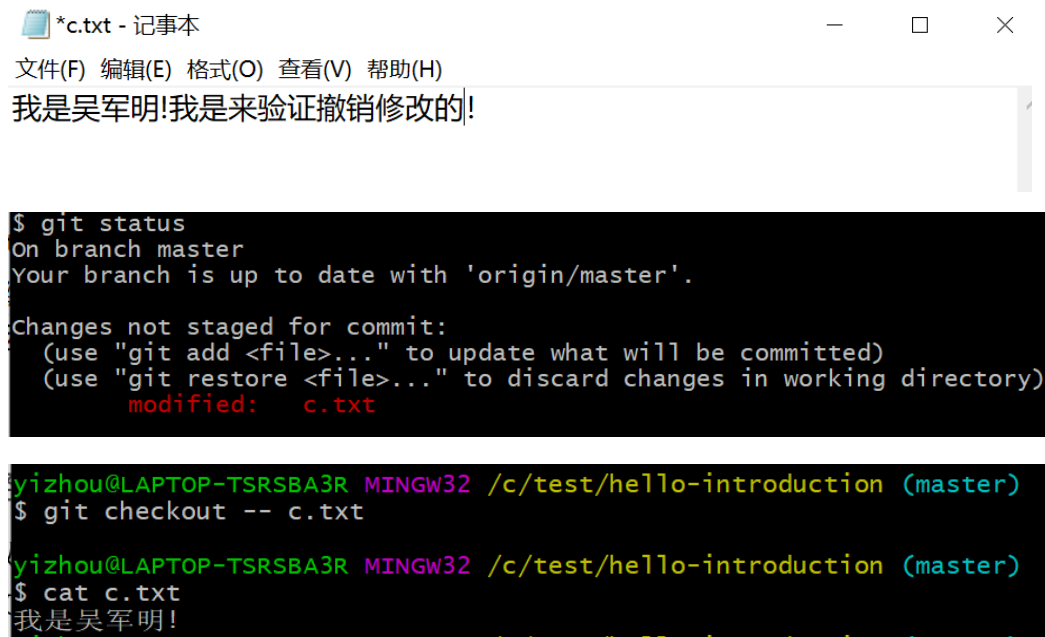
如果在使用“`git commit`”提交到本地仓库区后，再对文件进行修改时发现修改错误时，要使用 `git` 放弃本地的修改时，分为以下三种情况。

a) 未使用 `git add` 缓存修改时

可以使用 `git checkout -- filepathname` (比如：`git checkout -- c.txt`，不要忘记中间的“`--`”，不写就成了切换分支了!!)。放弃所有的文件修改可以使用 `git checkout .` 命令。此命令用来放弃掉所有还没有加入到缓存区（就是 `git add` 命令）的修改：内容修改与整个文件删除。但是此命令不会删掉新建的文件。

我们之前在本地仓库创建的 `c.txt` 文件，使用“`git commit`”提交代码到了本地仓库区，

之后我们发现还需要对该文件做一些修改，并且修改了此文件，见下图。这时我们还没使用 `git add` 缓存代码，这时可以使用 `git checkout -- c.txt` 撤销本地刚刚所做的修改。



The image shows a Notepad window titled "*c.txt - 记事本" with the text "我是吴军明!我是来验证撤销修改的!". Below it is a terminal window showing the following commands and output:

```
$ git status
On branch master
Your branch is up to date with 'origin/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:   c.txt

yizhou@LAPTOP-TSRSBA3R MINGW32 /c/test/hello-introduction (master)
$ git checkout -- c.txt

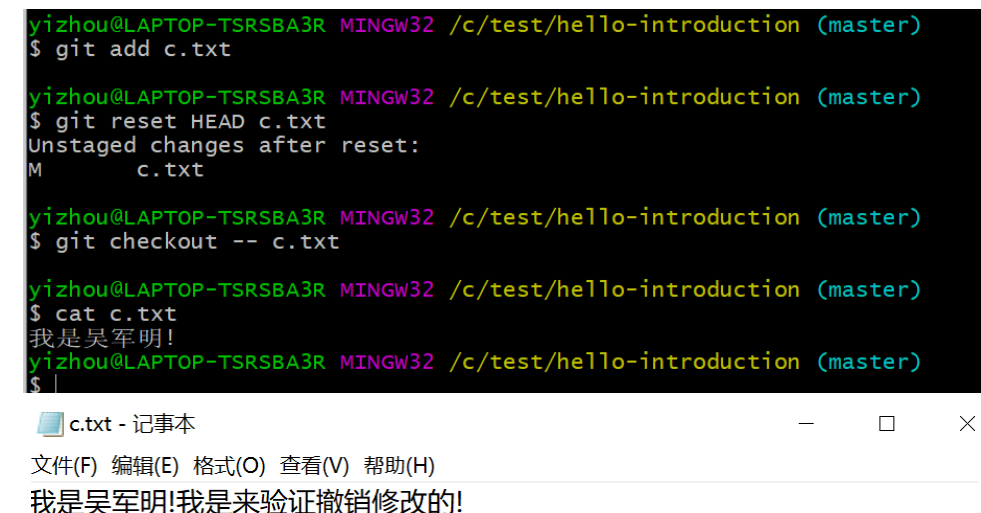
yizhou@LAPTOP-TSRSBA3R MINGW32 /c/test/hello-introduction (master)
$ cat c.txt
我是吴军明!
```

可以看到，我们使用 `git checkout -- c.txt` 后，`c.txt` 所做的修改已经被撤销了，由“我是吴军明!我是来验证撤销的!”变成了“我是吴军明!”。

b) 已经使用了 `git add` 缓存了代码

可以使用 `git reset HEAD filepathname`（比如：`git reset HEAD readme.md`）来放弃指定文件的缓存，放弃所有的缓存可以使用 `git reset HEAD .` 命令。

此命令用来清除 `git` 对于文件修改的缓存。相当于撤销 `git add` 命令所在的工作，即清空缓存区/暂存区。在使用本命令后，本地的修改并不会消失，而是回到了 a) 的状态。继续用 a) 中的操作，就可以放弃本地的修改。



The image shows a terminal window with the following commands and output:

```
yizhou@LAPTOP-TSRSBA3R MINGW32 /c/test/hello-introduction (master)
$ git add c.txt

yizhou@LAPTOP-TSRSBA3R MINGW32 /c/test/hello-introduction (master)
$ git reset HEAD c.txt
Unstaged changes after reset:
M    c.txt

yizhou@LAPTOP-TSRSBA3R MINGW32 /c/test/hello-introduction (master)
$ git checkout -- c.txt

yizhou@LAPTOP-TSRSBA3R MINGW32 /c/test/hello-introduction (master)
$ cat c.txt
我是吴军明!
yizhou@LAPTOP-TSRSBA3R MINGW32 /c/test/hello-introduction (master)
$
```

Below the terminal is a Notepad window titled "c.txt - 记事本" with the text "我是吴军明!我是来验证撤销修改的!".



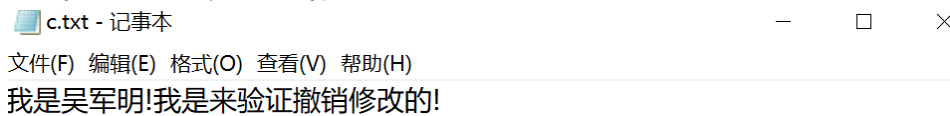
注意：git commit 提交到代码到仓库区时，使用 git checkout - “文件名”撤销本地修改都是指的是对工作区的文件进行了回退操作。而仓库区该文件的内容未发生变化，当使用 git push 指令把代码推送到远端时，推的还是修改之后的文件内容。要推送回退后的文件时，还得使用再次使用 git add 和 git commit 指令再次提交。

c) 已经用 git commit 提交了代码

可以使用 git reset --hard HEAD^ 来回退到上一次 commit 的状态。此命令可以用来回退到任意版本：git reset --hard commitid(提交的 ID 号)

你可以使用 git log 命令来查看 git 的提交历史。git log 的输出如下，这里可以看到的第一行就是最近的一次 commit 记录。红色圈起来的的就是 commitid。

1) 第一次提交的文件内容



2) 第二次修改并提交的文件内容



```
yizhou@LAPTOP-TSR5BA3R MINGW32 /c/test/hello-introduction (master)
$ git log
commit 535d88143cea392beda6e7d22f1ef23b4fc86fa1 (HEAD -> master)
Author: zhouyi <2283240768@qq.com>
Date: Sat Aug 8 20:13:02 2020 +0800

    deleted version

commit fe5c67113f1f384d29576df76ef529d9181eb16e
Author: zhouyi <2283240768@qq.com>
Date: Sat Aug 8 20:11:49 2020 +0800

    new add c.txt
```

那么我们可以通过 git reset --hard HEAD^ 回退到修改之前的版本。

```
yizhou@LAPTOP-TSR5BA3R MINGW32 /c/test/hello-introduction (master)
$ git reset --hard HEAD^
HEAD is now at fe5c671 new add c.txt

yizhou@LAPTOP-TSR5BA3R MINGW32 /c/test/hello-introduction (master)
$ cat c.txt
我是吴军明!我是来验证撤销修改的!
```

还可以通过 git reset --hard 535d88143cea392beda6e7d22f1ef23b4fc86fa1(提交版本号)

回退到你想要的任何一个版本上。

```
yizhou@LAPTOP-TSRSBA3R MINGW32 /c/test/hello-introduction (master)
$ git reset --hard 535d88143cea392beda6e7d22f1ef23b4fc86fa1
HEAD is now at 535d881 deleted version

yizhou@LAPTOP-TSRSBA3R MINGW32 /c/test/hello-introduction (master)
$ cat c.txt
我是吴军明!
```

读到这里，你已经掌握了 90% 的 git 基础知识，这些经常用到指令得常常练习，熟能生巧。另外还有其他的一些指令可去官网查询。