

# 1.Git配置

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。

这些环境变量，决定了 Git 在各个环节的具体工作方式和行为。这些变量可以存放在以下三个不同的地方：

- `/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。
- `~/.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。
- 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。

在 Windows 系统上，Git 会找寻用户主目录下的 `.gitconfig` 文件。主目录即 `$HOME` 变量指定的目录，一般都是 `C:\Documents and Settings\USER`。

此外，Git 还会尝试找寻 `/etc/gitconfig` 文件，只不过看当初 Git 装在什么目录，就以此作为根目录来定位。

## 1.1 用户信息

配置个人的用户名称和电子邮件地址：

```
$ git config --global user.name "cho.kyoku"
$ git config --global user.email "zhaoxu8673@gmail.com"
```

# 2.远程仓库

## 2.1 克隆远程仓库

如果想将远程仓库克隆下来，需要使用 `git clone` 命令

```
git clone https://github.com/ChoKyoku/Doc.git
```

<https://github.com/ChoKyoku/Doc.git> 就是远程仓库的地址

## 2.2 查看远程分支

使用 `git remote` 命令可以查看远程仓库，`origin`表示远程主机

使用`git remote -v` 命令可以查看远程仓库详细信息，包括远程仓库的地址

```
MINGW64:/c/Git/Doc

x-zhao@DESKTOP-PC7S3SF MINGW64 /c/Git/Doc (main)
$ pwd
/c/Git/Doc

x-zhao@DESKTOP-PC7S3SF MINGW64 /c/Git/Doc (main)
$ git remote
origin

x-zhao@DESKTOP-PC7S3SF MINGW64 /c/Git/Doc (main)
$ |
```

```
MINGW64:/c/Git/Doc

x-zhao@DESKTOP-PC7S3SF MINGW64 /c/Git/Doc (main)
$ pwd
/c/Git/Doc

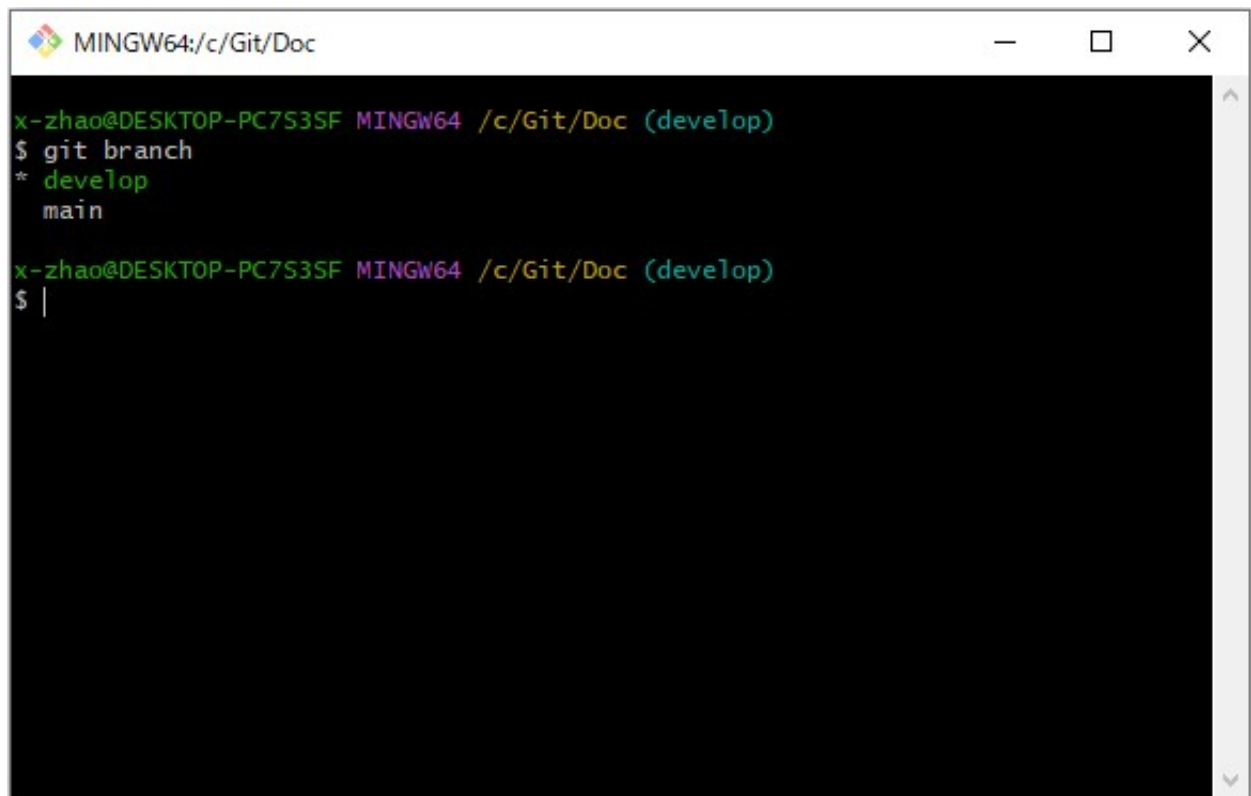
x-zhao@DESKTOP-PC7S3SF MINGW64 /c/Git/Doc (main)
$ git remote -v
origin https://github.com/ChoKyoku/Doc.git (fetch)
origin https://github.com/ChoKyoku/Doc.git (push)

x-zhao@DESKTOP-PC7S3SF MINGW64 /c/Git/Doc (main)
$
```

## 2.3 Git分支管理

### 2.3.1 列出分支

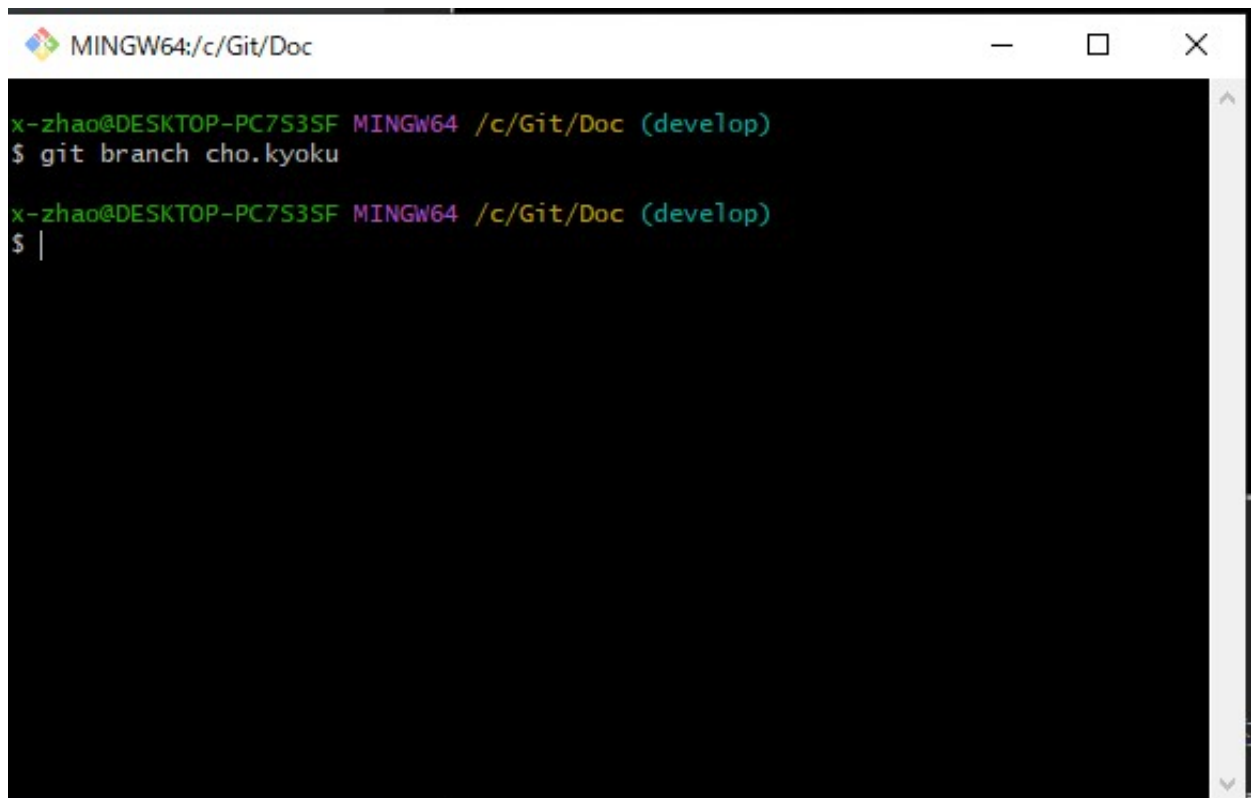
在git中，使用**git branch** 命令列出所有分支，没有参数的时候就列出本地所有的分支

A terminal window titled 'MINGW64:/c/Git/Doc' with standard window controls. The prompt is 'x-zhao@DESKTOP-PC7S3SF MINGW64 /c/Git/Doc (develop)'. The command '\$ git branch' is entered, and the output shows two branches: '\* develop' and 'main'. The prompt then changes to 'x-zhao@DESKTOP-PC7S3SF MINGW64 /c/Git/Doc (develop)' and a new line with '\$ |' is shown.

```
MINGW64:/c/Git/Doc
x-zhao@DESKTOP-PC7S3SF MINGW64 /c/Git/Doc (develop)
$ git branch
* develop
  main
x-zhao@DESKTOP-PC7S3SF MINGW64 /c/Git/Doc (develop)
$ |
```

### 2.3.2 创建分支

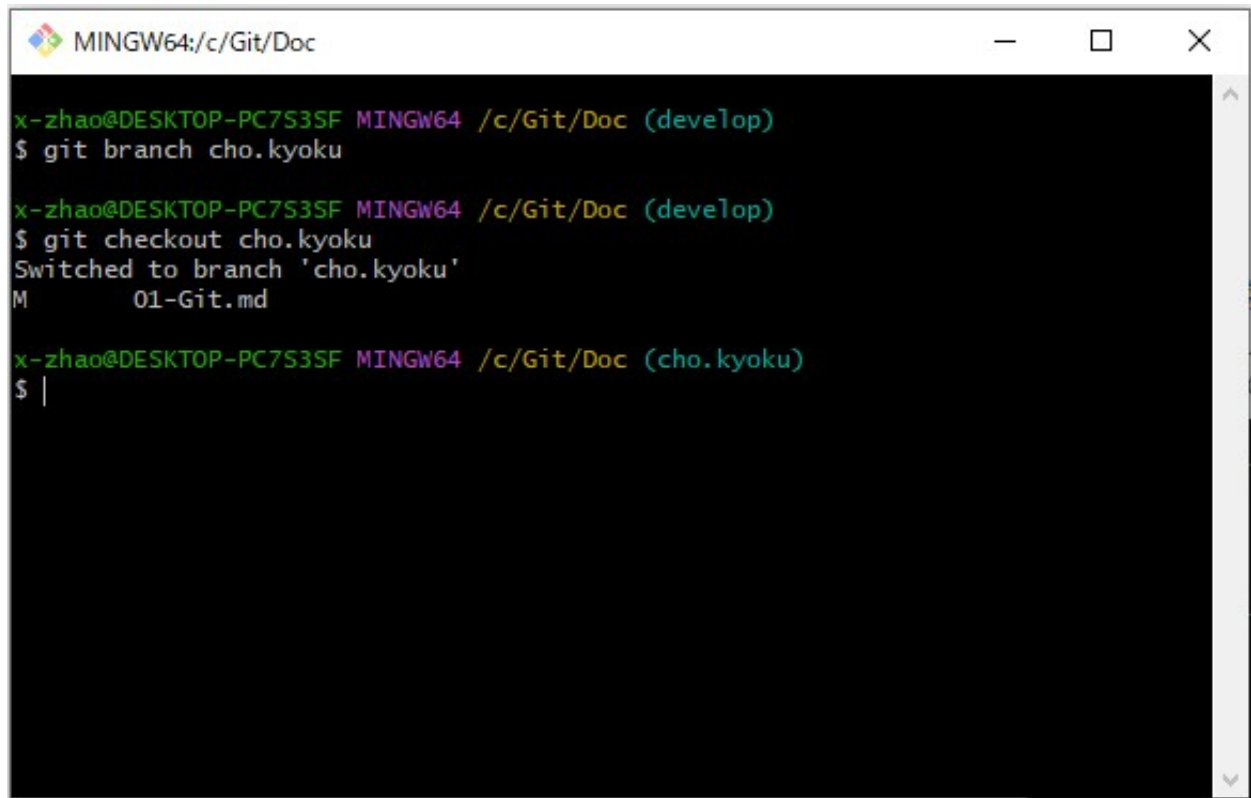
在git中，使用 **git branch 分支名** 的方式来创建新的分支。

A terminal window titled 'MINGW64:/c/Git/Doc' with standard window controls. The prompt is 'x-zhao@DESKTOP-PC7S3SF MINGW64 /c/Git/Doc (develop)'. The command '\$ git branch cho.kyoku' is entered. The prompt then changes to 'x-zhao@DESKTOP-PC7S3SF MINGW64 /c/Git/Doc (develop)' and a new line with '\$ |' is shown.

```
MINGW64:/c/Git/Doc
x-zhao@DESKTOP-PC7S3SF MINGW64 /c/Git/Doc (develop)
$ git branch cho.kyoku
x-zhao@DESKTOP-PC7S3SF MINGW64 /c/Git/Doc (develop)
$ |
```

### 2.3.3 切换分支

使用 **git checkout 分支名** 的方式来切换到想要的分支。切换好新的分支之后，所有的内容也全部都更换成新的分支的内容了。

A terminal window titled 'MINGW64:/c/Git/Doc' with standard window controls. The prompt is 'x-zhao@DESKTOP-PC7S3SF MINGW64 /c/Git/Doc (develop)'. The user enters '\$ git branch cho.kyoku'. The prompt changes to '(develop)'. The user enters '\$ git checkout cho.kyoku'. The terminal shows 'Switched to branch 'cho.kyoku'' and 'M 01-Git.md'. The prompt changes to '(cho.kyoku)'. The user enters '\$ |' and the cursor is on the next line.

```
MINGW64:/c/Git/Doc
x-zhao@DESKTOP-PC7S3SF MINGW64 /c/Git/Doc (develop)
$ git branch cho.kyoku

x-zhao@DESKTOP-PC7S3SF MINGW64 /c/Git/Doc (develop)
$ git checkout cho.kyoku
Switched to branch 'cho.kyoku'
M      01-Git.md

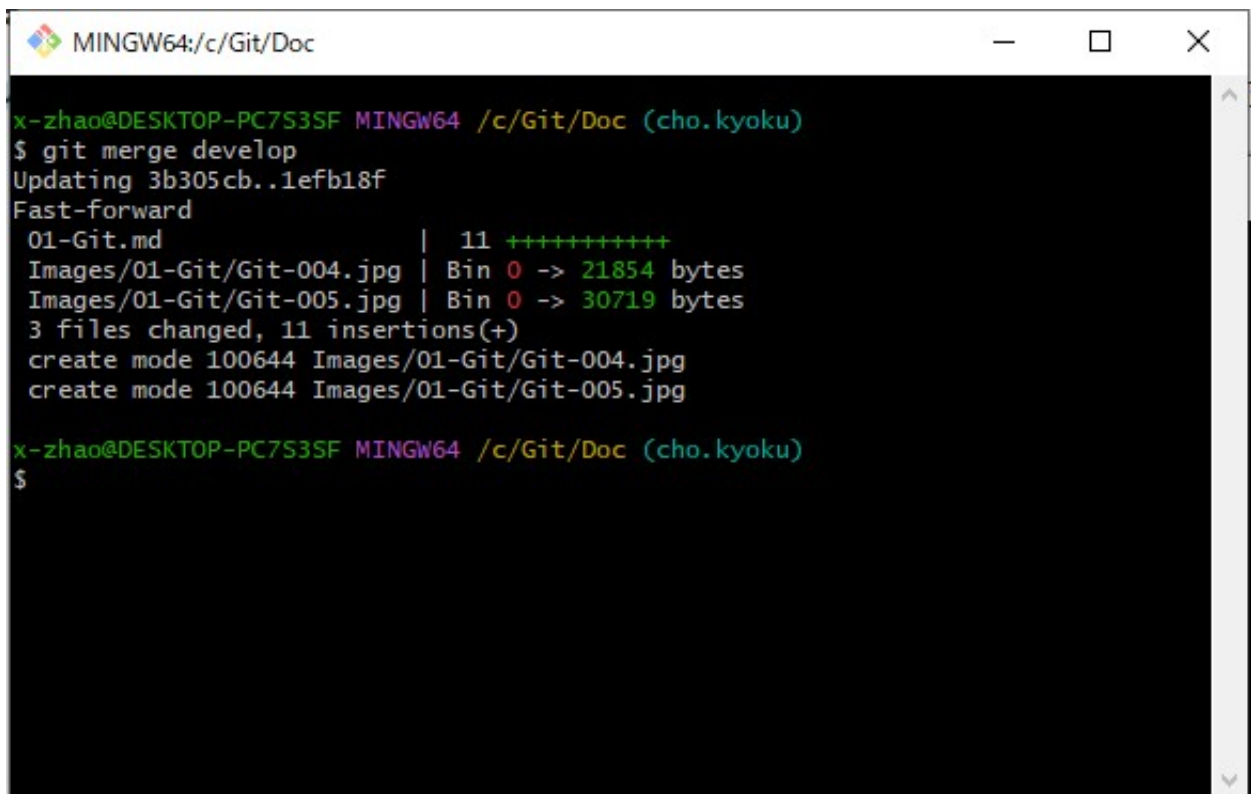
x-zhao@DESKTOP-PC7S3SF MINGW64 /c/Git/Doc (cho.kyoku)
$ |
```

### 2.3.4 合并分支

在git中，如果想将其他的分支内容合并到当前的分支中的话可以使用 **git merge** 命令

语法:git merge oBranch

作用:将oBranch的内容合并到当前分支中来

A terminal window titled 'MINGW64:/c/Git/Doc' with standard window controls. The prompt is 'x-zhao@DESKTOP-PC7S3SF MINGW64 /c/Git/Doc (cho.kyoku)'. The user enters '\$ git merge develop'. The terminal shows 'Updating 3b305cb..1efb18f' and 'Fast-forward'. It then shows a diff for '01-Git.md' with 11 insertions. Below that, it shows 'Images/01-Git/Git-004.jpg' and 'Images/01-Git/Git-005.jpg' being created from binary files. The prompt returns to '(cho.kyoku)'. The user enters '\$' and the cursor is on the next line.

```
MINGW64:/c/Git/Doc
x-zhao@DESKTOP-PC7S3SF MINGW64 /c/Git/Doc (cho.kyoku)
$ git merge develop
Updating 3b305cb..1efb18f
Fast-forward
 01-Git.md | 11 ++++++++
Images/01-Git/Git-004.jpg | Bin 0 -> 21854 bytes
Images/01-Git/Git-005.jpg | Bin 0 -> 30719 bytes
3 files changed, 11 insertions(+)
create mode 100644 Images/01-Git/Git-004.jpg
create mode 100644 Images/01-Git/Git-005.jpg

x-zhao@DESKTOP-PC7S3SF MINGW64 /c/Git/Doc (cho.kyoku)
$
```

## 3.常用操作

### 3.1 git clone

用于将远程仓库的内容克隆到本地

语法: `git clone [url]`

### 3.2 git add

可将文件添加到暂存区(一个或多个文件)

语法: `git add [file1] [file2] ...`

`git add -A` : 用于将当前目录下的所有文件都添加到暂存区

### 3.3 git commit

将暂存区中的内容添加到本地仓库中

语法: `git commit -m [message]`

message : 可以是一些备注信息

`git commit -a` : 执行该指令时, 修改文件后不需要执行`git add`命令, 直接来提交

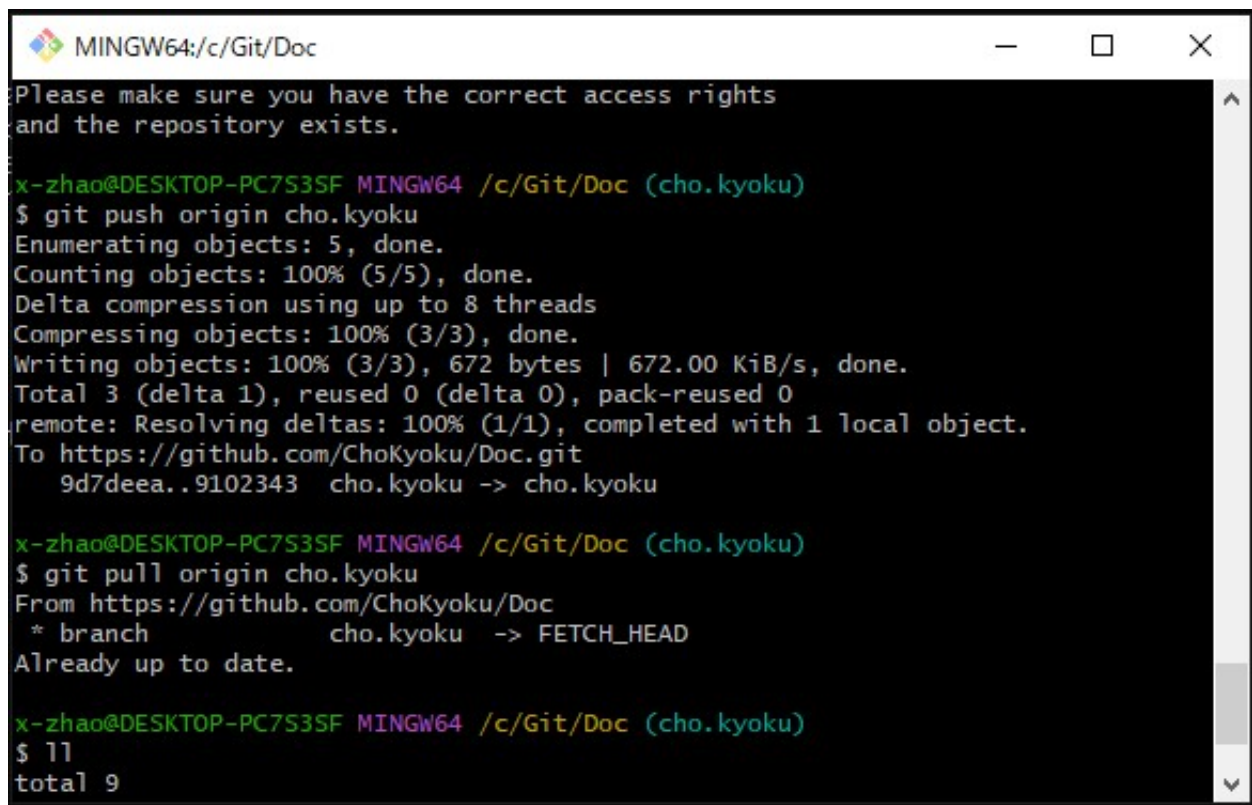
### 3.4 git push

用于将本地仓库中的内容推送到远程仓库中

语法: `git push <远程仓库地址>:<本地分支> <远程分支>`

如果本地分支和远程分支名称一样的话可以省略本地分支名

`git push origin cho.kyoku`



```
MINGW64:/c/Git/Doc
Please make sure you have the correct access rights
and the repository exists.

x-zhao@DESKTOP-PC7S3SF MINGW64 /c/Git/Doc (cho.kyoku)
$ git push origin cho.kyoku
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 672 bytes | 672.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/ChoKyoku/Doc.git
   9d7deea..9102343  cho.kyoku -> cho.kyoku

x-zhao@DESKTOP-PC7S3SF MINGW64 /c/Git/Doc (cho.kyoku)
$ git pull origin cho.kyoku
From https://github.com/ChoKyoku/Doc
 * branch                cho.kyoku -> FETCH_HEAD
Already up to date.

x-zhao@DESKTOP-PC7S3SF MINGW64 /c/Git/Doc (cho.kyoku)
$ ll
total 9
```

## 3.5 git checkout

## 3.6 git pull

用于从过程获取代码并合并到本地版本中

git pull 实际是git fetch 和 git merge FETCH\_HEAD的缩写

语法: git pull <过程主机名> <远程分支名> : <本地分支名>

git pull

git pull origin

git pull origin master : cho.kyoku

如果远程分支和当前分支合并的话，则冒号后面的内容就可以省略

git pull origin cho.kyoku

## 3.7 git restore

git store 命令主要用于恢复文件到

### 1.恢复已修改但未暂存（未git add）的文件

如果修改了文件，但还没有使用 git add，则可以使用下面指令恢复到上次提交的版本

```
git restore <文件名>
```

或恢复所有为未暂存的修改

```
git restore .
```

### 2.恢复已暂存（git add）但未提交的文件

```
git restore --staged <FileName>
```

如果同时还想恢复到工作区的上次提交状态

```
git restore --staged <文件名>  
git restore <文件名>
```

### 3.恢复已提交的内容

#### 3.1 恢复到上次提交的状态

如果你已经提交了更改，但想回到上一次提交的状态，可以使用以下命令：

```
git reset --hard HEAD~1
```

#### 3.2 恢复单个文件到某次提交

可以恢复某个文件到指定的提交版本

```
git checkout <提交哈希值> -- <文件名>
```

在新的版本中，可以使用以下版本代替

```
git restore --source=<提交哈希值> <文件名>
```