Git 使用手册整理

# 1.安装

Windows – 推荐使用[git for windows](https://git-for-windows.github.io/)，它包括了图形工具以及命令行模拟器

地址：<https://git-for-windows.github.io/>

# 2.配置

安装完git，首要任务是做一些简单的配置，最重要的是用户名及邮箱，打开终端，执行以下命令。

$ **git config --global** user.name "My Name"

$ **git config --global** user.email myEmail@example.com

配置好这两项，Git就能记录下来是谁做的动作，一切都更有组织性了。

# 3.创建一个新仓库 git init

Git会把所有文件以及历史记录直接记录成一个文件夹保存在你的项目中。创建一个新的仓库，首先要去到项目路径下，执行git init。这时Git会创建一个隐藏的文件夹 .git，所有的历史和配置信息都储存在其中。

比如我们在桌面创建一个文件夹 test\_repo, 打开终端，输入：

$ cd test\_repo/

$ **git init**

执行命令后出下面这行说明我们的仓库建立好了，但现在是控的，试着新建一个hello.txt文本文件到这个文件夹里。

Initialized empty Git repository in C:/Users/phh/Desktop/test\_repo/.git/

# 4.检查状态 git status

git status 是一个非常重要的命令，它反馈给我们仓库当前状态的信息：是否为最新代码，有什么更新等等。在我们新建的仓库中执行git status会得到以下内容：

phh@DESKTOP-10D0E5N MINGW64 ~/Desktop/test\_repo (master)

$ **git status**

On branch master

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

hello.txt

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

反馈信息告诉我们，hello.txt尚未跟踪，这是说这个文件是新的，git不知道是应该跟踪它的变动还是直接忽略。为了跟踪我们的新文件，我们需要暂存它。

# 5.暂存 git add

Git有个概念叫“暂存区“，你可以把它看成一块空白的画布，包裹着所有你可能会提交的变动。它一开始是空的，可以通过 git add 命令添加内容，最后使用 git commit 提交（创建一个快照）。

这个例子中只有一个文件，让我们先add它：

$ **git add** hello.txt

如果需要提交目录下的所有内容，可以这样做：

$ **git add -A**

再次使用git status查看状态试试：

$ **git status**

On branch master

No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: hello.txt

我们的文件已经准备好可以提交了。状态信息还告诉我们暂存区文件发生了什么变动，这里我们新增了一个文件，同样可以做修改和删除。取决于我们在上一次git add之后发生了什么。

# 6.提交 git commit

一次提交代表着我们的仓库到了一个新的状态，就像是一个快照，允许我们像使用时光机一样回到之前的某个时间点。

创建提交，需要我们至少在到暂存区有一次修改（刚才我们做了git add），然后输入命令：

$ **git commit -m "这是提交说明信息"**

[master (root-commit) 38a7734] 这是提交说明信息

1 file changed, 1 insertion(+)

create mode 100644 hello.txt

这就创建了一次从暂存区的提交（加入hello.txt），**-m "这是提交说明信息"** 是用户对这次提交的描述，建议写成有意义的描述性信息。

# 7.远程仓库

## 1):链接远程仓库 git remote add

为了能够上传到远程仓库，我们需要先建起链接。这里我们的示例远程仓库地址为：

<https://github.com/phh2008/hello-world.git>，请自行尝试在github，或者gitlab创建远程仓库。

把本地仓库链接到github上，执行以下命令：

$ **git remote add** **origin** https://github.com/phh2008/hello-world.git

一个项目可以同时拥有多个远程仓库，为了区分通常会起不同的名字。通常主要的远程仓库被称为origin。

## 2):上传到远程仓库 git push

把本地的提交传送到服务器的动作叫做push。每次我们要提交修改到服务器上时，都会使用到git push。

命令用法：git push 远程仓库名字 分支名字

$ **git push** **origin master**

Counting objects: 100% (3/3), done.

Writing objects: 100% (3/3), 249 bytes | 249.00 KiB/s, done.

Total 3 (delta 0), reused 0 (delta 0)

To https://github.com/phh2008/hello-world.git

\* [new branch] master -> master

取决于你使用的服务器，push过程中你可能需要验证身份（输入用户名、密码，请先在网站上进行注册）。如果没有出差错，现在用浏览器看你的远程仓库上已经有hello.txt了。

## 3):克隆仓库 git clone

我们可以用git clone 命令把远程仓库下载到本地。比如下载上一步上传的项目：

$ **git clone** **https://github.com/phh2008/hello-world.git**

Cloning into 'hello-world'...

remote: Enumerating objects: 3, done.

remote: Counting objects: 100% (3/3), done.

remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0

Unpacking objects: 100% (3/3), done.

## 4):获取远程仓库的更新 git pull

如果多人协作一个项目，其他人更新了远程仓库上的代码，那么我可以通过git pull 来获取有更新的内容：命令用法 git pull <远程仓库名称> <分支名称>

$ **git pull origin master**

remote: Enumerating objects: 5, done.

remote: Counting objects: 100% (5/5), done.

remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0

Unpacking objects: 100% (3/3), done.

From https://github.com/phh2008/hello-world

\* branch master -> FETCH\_HEAD

38a7734..d742def master -> origin/master

Updating 38a7734..d742def

Fast-forward

hello.txt | 2 ++

1 file changed, 2 insertions(+)

# 8.分支管理

当你在做一个新功能的时候，最好是在一个独立的区域上开发（原始项目的拷贝），通常称之为分支。分支之间相互独立，并且拥有自己的历史记录，直到你决定把他们合并到一起。这样做的原因是：

已经可以运行的稳定版本的代码不会被破坏

不同的功能可以由不同开发者同时开发

开发者可以专注于自己的分支，不用担心被其他人破坏

在不确定哪个版本更好之前，同一个特性可以在不同的分支上创建多个版本，便于比较

## 1):新建分支 git brach

每一个仓库的默认分支都叫master, 创建新分支可以用git branch <name>命令：

$ **git branch** **dev**

上面命令创建了一个名为 dev 的新分支，它和当前分支(master)内容是一样的。

## 2):切换分支 git checkout

使用git brach 可以查看有那些分支：

**$ git branch**

dev

\* master

\* 号表示当前所在分支为master ，现在我们想在新分支 dev 上开发新功能，

使用git check切换分支。用法： git checkout <分支名称>

**$ git checkout** **dev**

Switched to branch 'dev'

## 3):合并分支 git merge

新分支任务开发完成，现在想要把这些更新合并到master 主分支，只有合并后主分支才能看到更新。命令使用：git merge <来源分支>

示例合并步骤：比如dev并合到master

A：切到需要被合并到的分支 master

$ **git checkout** **master**

Switched to branch 'master'

Your branch is up to date with 'origin/master'.

B：执行合并命令

$ **git merge** **dev**

Updating d742def..4d00746

Fast-forward

hello.txt | 2 ++

1 file changed, 2 insertions(+)

C：删除分支

分支合并完后，如果不想保留，可以删除分支，使用命令：git brach -d <分支名>

# 9.撤销修改与回退版本

## 1):撤销修改

撤销修改是指将工作区中的修改撤销。

使用：git checkout <文件名或路径>

A：撤销工作区中当前目录的所有更改

$ **git checkout .**

Updated 1 path from the index

B：撤销工作区中指定文件的修改

$ **git checkout -- hello.txt**

注意：Git checkout 命令只能销工作区中的修改，而不能撤销暂存区中的修改（即add/commit的不适用）

## 2):版本回退

版本回退是指将版本库从当前版本回退到其他的版本

使用：git reset --<选项> <版本号>

在 Git 中，HEAD 指针指向的是当前版本，也就是最新的 commit id，上一个版本是 HEAD^，上上一个版本就是 HEAD^^，上50个版本可以写成 HEAD~50。

可以使用下面的命令，将 Git 回退到上一个版本：

$ **git reset --hard HEAD^**

也可使用git log 查看提交日志，获取提交commit id

git reset --hard <commit id> 比如：

$ **git reset --hard** d742def

HEAD is now at d742def Update hello.txt

版本号(commit id) 一般会很长，可不用写全，一般写前七位够了，git自动匹配

Git reset 的常用选项（如上面示例中的--hard）：

|  |  |
| --- | --- |
| --mixed | reset HEAD and index。表示重置 HEAD 指针和 index 暂存区，但保持工作区不变。它是默认选项 |
| --soft | reset only HEAD。表示仅仅重置 HEAD 指针，即只改变 HEAD 的指向，但保持工作区和暂存区不变 |
| --hard | reset HEAD, index and working tree。表示重置 HEAD 指针、index 暂存区和工作区。这个才是完整的版本回退 |

# 10.解决冲突

冲突经常出现在合并分支或者是拉取别人的代码。有些时候git能自动处理冲突，其他时候需要我们手动处理。

可以调用外部GUI冲突解决工具：**git mergetool**

比如：整合Beyond Compare，配置如下

git config --global diff.tool bc4

git config --global difftool.prompt false

git config --global difftool.bc4.cmd '"E:\dev\Beyond Compare 4\BCompare.exe" "$LOCAL" "$REMOTE"'

git config --global merge.tool bc4

git config --global mergetool.prompt false

git config --global mergetool.bc4.cmd '"E:\dev\Beyond Compare 4\BCompare.exe" "$LOCAL" "$REMOTE" "$BASE" "$MERGED"'

git config --global mergetool.bc4.trustexitcode true

# 11. gitignore 文件配置

大部分项目中，会有些文件、文件夹是我们不想提交的。为了防止使用git add -A时不小心提交，我们可以利用 .gitignore 文件：

A：在项目根目录创建 .gitignore文件

B：在文件中列出不需要提交的文件名、文件夹名，每个一行

C：.gitignore文件需要像普通文件一样add、commit和push

示例：

\*.log

build/

node\_modules/

.idea/

my\_notes.txt

“/”说明是一个文件夹，里面的所有内容都被递归忽略

# 12.图形化工具

1：SourceTree

2：Idea

# 13.更多帮助

详细使用文档：<https://git-scm.com/book/zh/v2>