Git 教程总结

简介：Git 是一个开源的分布式版本控制系统，用于敏捷高效的处理或大或小的项目。Git 不仅仅是个版本控制系统，它也是个内容管理系统，工作管理系统等。

一、安装 Git：

1. 在 Windows 系统下安装 Git：

   可以直接从 Git 的官网直接下载（https://git-scm.com/），安装完成后，再开始菜单找到 ”Git” -> ”Git Bash” 打开，如果弹出一个类似命令的窗口，就说明 Git 安装成功。

   安装完成后，还需进行用户名和邮箱设置（与你在 GitHub 上注册的一样），在命令行输入：

```
$ git config --global user.name "Your Name"

$ git config --global user.email "email@example.com"
```

2. 在 Mac 系统下安装 Git：

   第一种方法是安装 homebrew，然后通过 homebrew 安装 Git，具体方法请参考 homebrew 的文档：http://brew.sh/。

   第二种方法是直接从 App Store 安装 Xcode，Xcode 集成了 Git，不过默认没有安装，你需要运行 Xcode，选择菜单”Xcode” -> “Preferences”，在弹出窗口中找到”Downloads”，选择”Command Line Tools”，点”install”就可以完成安装了。

3. 在 Linux 系统下安装 Git：

   首先，你可以输入 git，看系统是否安装 Git。

如果你碰巧使用 Debian 或 Ubuntu Linux，通过一条”sudo apt-get install git”就可以直接完成 Git 的安装。

```
$ git

The program 'git' is currently not installed. You can install it by typing:

sudo apt-get install git
```

二、创建版本库：

版本库(repository)，这个目录里面的所有文件都可以通过 Git 来管理，每个文件的修改和删除都能跟踪，以便在将来某个时刻可以“还原”。

1. 创建：
   首先，选择一个合适的地方，创建一个空目录（mkdir 用于创建一个子目录，pwd 用于显示当前目录）：

```
$ mkdir learngit

$ cd learngit

$ pwd

/Users/michael/learngit
```

第二步，通过“git init”命令把这个目录变成 Git 可以管理的仓库：

```
$ git init

Initialized empty Git repository in /Users/michael/learngit/.git/
```

2. 把文件添加到版本库：

现在，在创建的 learngit 目录下，编写一个 readme.txt 文件。

第一步：用”git add”命令将文件添加到缓存（接着可以用”git status”命令查看项目的当前状态）。

```
$ git add readme.txt
```

第二步：用”git commit”命令提交文件到仓库。

```
$ git commit -m "wrote a readme file"

[master (root-commit) eaadf4e] wrote a readme file

 1 file changed, 2 insertions(+)

 create mode 100644 readme.txt
```

3. 修改：

使用”git status”命令查看本地仓库的状态，查看文件是否被修改。

如果想看具体修改了什么内容，可以使用”git diff”命令比较两次修改的差异。

了解修改的内容后，使用”git add”命令提交，在执行”git commit”之前，最好使用”git status”查看状态。

三、版本回退：

版本控制系统有一个命令”git log（git log –pretty=online）”可以查看历史记录，它显示从近到远提交的日志信息，以便确定要回退到哪个版本。

如果想回退上一个版本，使用”git reset”命令。

```
$ git reset --hard HEAD^

HEAD is now at e475afc add distributed
```

然后使用”git log”查看现在版本库的状态。

小结：

HEAD 指向的版本就是当前版本，因此，Git 允许我们在版本的历史之间穿梭，使用命令”git reset –hard commit_id”。

穿梭（回退）前，用”git log”可以查看提交历史，以便确定要回退到哪个版本。

要重返未来，用”git log”可以查看命令历史，以便确定要回到未来的哪个版本。

四、工作区和暂存区：

工作区：就是在电脑文件夹里能够看到的目录，例如前面的 learngit 文件夹就是一个工作区。

版本库：

工作区有一个隐藏目录.git，它不是一个工作区，而是 Git 的版本库。

版本库里包括称为 stage（或者叫 index）的暂存区，还包括各种分支，如：master（Git 自动创建的第一个分支）。

“git add”：把文件修改添加到暂存区。

“git commit”：把暂存区的所有内容提交到当前分支。

五、撤销修改：

“git checkout --file”可以撤销工作区的修改：

```
$ git checkout -- readme.txt
```

命令”git checkout –readme.txt”意思就是，把 readme.txt 文件在工作区的修改全部撤销，让这个文件回到最近一次”git add”或”git commit”时的状态。

六、删除文件：

一种情况是确定要从版本库中删除一个文件，那就用”git rm”删掉，并且用”git commit”提交。

```
$ git rm test.txt

rm 'test.txt'



$ git commit -m "remove test.txt"

[master d46f35e] remove test.txt

 1 file changed, 1 deletion(-)

 delete mode 100644 test.txt
```

另一种情况是删错了，因为版本库里还有，可以很轻松的把删除的文件恢复到最新版本。

```
$ git checkout -- test.txt
```

七、远程仓库：

1. 添加远程仓库：

好处：在 GitHub 上创建一个 Git 仓库，可以让它与本地仓库进行同步，这样，GitHub 上的仓库既可作为备份，又可让其他人通过该仓库来协作。

创建远程仓库，我们需登录 GitHub，在 GitHub 右上角找到”create a new repo”按钮，点击创建。

如果是将本地仓库直接导入到远程仓库中，远程仓库没有任何分支和代码，只需要执行如下命令：

```
$ git remote add origin https://github.com/haijingli/html5.git

$ git push -u origin master
```

第一次 push 的时候，加上-u 参数，Git 就会把本地的 master 分支进行关联起来，以后的 push 操作就不再需要加上-u 参数了。

2. 从远程克隆：

远程库准备好了以后，下一步可以用命令”git clone”克隆一个本地库。

```
$ git clone git@github.com:michaelliao/gitskills.git

Cloning into 'gitskills'...

remote: Counting objects: 3, done.

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

Receiving objects: 100% (3/3), done.
```

小结：

要克隆一个仓库，首先必须知道仓库的地址，然后使用”git clone”命令克隆。

Git 支持多种协议，包括 http，但通过 ssh 支持的原生 git 的协议速度快。

八、分支管理：

1. 创建与合并分支：

Git 的分支可以让你在主线（master）之外进行代码提交，同时又不会影响代码库的主线。

分支的作用体现在多人协作开发中，把自己需要实现的功能代码提交到一个分支中，其他同事可以继续使用主分支进行开发，不会对他们的代码造成影响，当你完成功能，测试通过后再把你的功能分支合并到主线。

小结：

查看分支：git branch

创建分支：git branch <name>

切换分支：git checkout <name>

创建+切换分支：git checkout -b <name>

合并某分支到当前分支：git merge <name>

删除分支：git branch -d <name>

2. stash 存入：

当我们工作还没完成，不能 commit 时，可以用”git stash”将现场工作存储起来：

```
$ git stash
Saved working directory and index state WIP on dev: f52c633 add merge
```

而此时就和撤销了所以的修改和 add 一样，这些都被暂存起来了，用命令”git stash list”可以查看存储起来的工作现场。

3. bug 分支：

把工作现场存好之后，就能放心的到处切换分支了。比如在 dev 分支上修复一个 bug，就需要切换到这个分支：

```
$ git checkout dev
```

然后在此基础上创建一个 bug 分支（就是普通的分支），分支名尽量体现 bug（如编号 bug）：

```
$ git checkout -b bug-101
```

接着修复 bug，提交一下：

```
$ git add xxx
$ git commit -m "fix bug 101"
[issue-101 4c805e2] fix bug 101
 1 file changed, 1 insertion(+), 1 deletion(-)
```

修复完成后，切换到 dev 分支，并完成合并：

```
$ git checkout dev
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 6 commits.
  (use "git push" to publish your local commits)

$ git merge --no-ff -m "merged bug fix 101" issue-101
Merge made by the 'recursive' strategy.
 readme.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

最后进行删除 bug 分支：

```
$ git branch -d bug-101
```

4. feature 分支：

当添加一个新功能时，最好新进一个 feature 分支。

当开发 feature 时也应该像 bug 那样新建一个分支，然后合并到要添加这个 feature 的分支上。

Bug 与 feature 使用上主要的区别在于，feature 可能因为需求的改变而取消，但 bug 几乎不会，要删除这样一个没有经过合并的分支，只要把-d 改成-D 即可强制删除。

5. stash 恢复：

切换回刚刚工作的分支上：

```
$ git checkout master
```

如果要恢复但不删除刚刚保存的 stash 记录，使用：

```
$ git stash apply
```

如果要恢复并删除刚刚保存的 stash 记录，使用：

```
$ git stash pop
```

九、标签管理：

标签类似于快照功能，可以给版本库打一个标签，记录某个时刻库的状态。也可以随时恢复到某时刻的状态。

1. 创建标签：

首先切换到需要打标签的分支上：

```
$ git branch

* dev

  master

$ git checkout master

Switched to branch 'master'
```

然后，敲击"git tag <name>"命令，就可以打一个新的标签。

```
$ git tag v1.0
```

可以用"git tag"命令查看所有标签。

```
$ git tag

v1.0
```

可以用"git show <tagnam>"命令查看标签的具体信息。

```
$ git show v0.9

commit f52c63349bc3c1593499807e5c8e972b82c8f286 (tag: v0.9)

Author: Michael Liao <askxuefeng@gmail.com>

Date:   Fri May 18 21:56:54 2018 +0800

    add merge

diff --git a/readme.txt b/readme.txt

...
```

还可创建带说明的标签，用-a 指定标签名，-m 指定说明文字。

```
$ git tag -a v0.1 -m "version 0.1 released" 1094adb
```

小结：

推送标签到远程，使用”git push origin <tagname>”命令。

推送全部未推送到远程的本地标签，使用”git push origin --tags”。

本地删除标签可以用”git tag -d <tagname>”命令。

删除一个远程标签，使用”git push origin :refs/tags/<tagname>”。

十、自定义 Git：

1. 让 Git 显示颜色，让输出命令更显目：

```
$ git config --global color.ui true
```

2. 忽略某些文件：忽略某些文件时，需要编写”.gitignore”文件；
3. ”.gitignore”文件本身要放到版本库里，并且可以对”.gitignore”做版本管理；
4. 如果想要简化指令代码，我们可以配置别名，例如为”git status”配置别名为”git st”：

```
$ git config --global alias.st status
```

5. 配置文件：每个仓库的配置文件都放在”.git/config”文件中，而当前用户的 Git 配置文件放在用户主目录下的一个隐藏文件”.gitconfig”中。

十一、搭建 Git 服务器：

第一步，安装 git:

```
$ sudo apt-get install git
```

第二步，创建一个 Git 用户，用来运行 Git 服务：

```
$ sudo adduser git
```

第三步，创建证书登录：

收集所有需要登录的用户公钥，就是他们自己的 id_rsa.pub 文件，把所有公钥导入到/home/git/.ssh/authorized keys 文件里，一行一个。

第四步，初始化 Git 仓库：

先选定一个目录作为 Git 仓库，假定/srv/sample.git，在/srv 目录下输入命令：

```
$ sudo git init --bare sample.git
```

第五步，禁止 shell 登录：

处于安全考虑，第二步创建的 git 用户不允许登录 shell，这可以通过编辑/etc/password 文件完成。找到类似下面的一行：

```
git:x:1001:1001:,,,:/home/git:/bin/bash
```

改为：

```
git:x:1001:1001:,,,:/home/git:/usr/bin/git-shell
```

这样 git 用户可以正常通过 ssh 使用 git，但无法登录 shell，因为我们为 git 用户指定的 git-shell 每次一登录就自动退出。

第六步，克隆远程仓库：

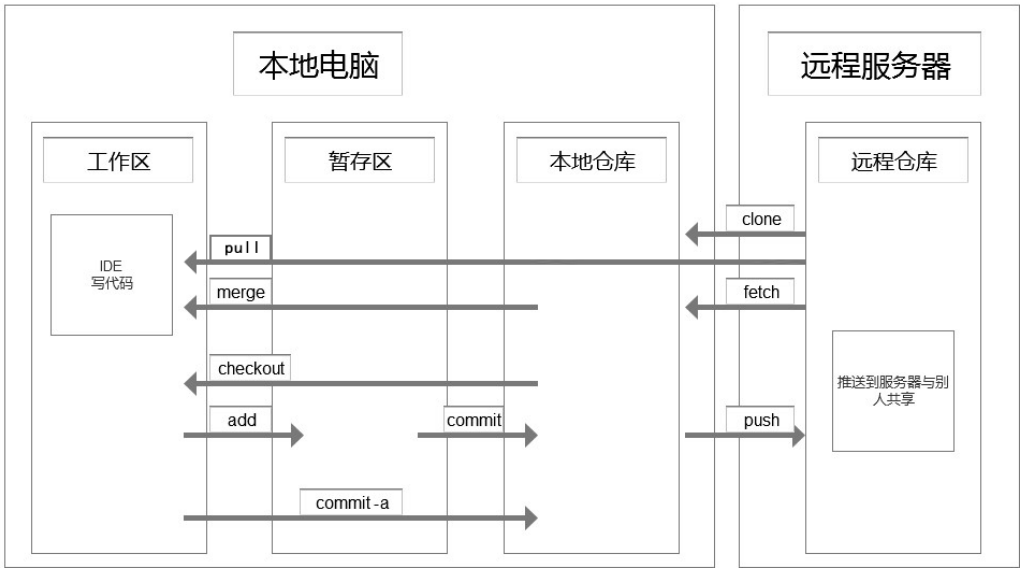
现在，可以通过”git clone”命令克隆远程仓库了，在各自的电脑上运行。

```
$ git clone git@server:/srv/sample.git

Cloning into 'sample'...

warning: You appear to have cloned an empty repository.
```

十二、总结：

git 工作流程：

1. 从远程仓库克隆 Git 作为本地仓库（git clone）。
2. 从本地仓库中 checkout（切换分支）代码，然后进行代码修改。
3. 在提交前先将代码提交到暂存区（git add）。
4. 提交修改，提交到本地仓库，本地仓库保存修改的各个历史版本（git commit）。
5. 在修改完后，需要和团队成员共享代码时，可以将代码 push 到远程仓库。



工作区：

工作区就是你在电脑能够看到的目录，比如我们创建的（learngit）文件夹。

版本库：

我们创建的（learngit）文件夹，这里面有个隐藏的".git"文件就是版本库。版本库里面包括被称为 stage（或者叫 index）的**暂存区**，还有 Git 为我们自动创建的第一个分支 master，以及指向 master 的一个指针叫 HEAD。

从远程仓库取代码：

1. git fetch：相当于从远程获取最新版本到本地，不会自动 merge（合并代码）。
2. git pull：相当于从远程获取最新版本并 merge 到本地。