

一、Git 简介：

1. Git (c 语言) linux
2. 集中式 vs 分布式

Git: 分布式版本控制系统根本没有“中央服务器”，每个人的电脑上都是一个完整的版本库。和集中式版本控制系统相比，分布式版本控制系统的安全性要高很多。

3. Git 安装
4. 创建版本库

```
master@LAPTOP-NPLOMRFC MINGW64 ~/learngit
$ cd /d/

master@LAPTOP-NPLOMRFC MINGW64 /d
$ mkdir learngit

master@LAPTOP-NPLOMRFC MINGW64 /d
$ cd learngit

master@LAPTOP-NPLOMRFC MINGW64 /d/learngit
$ git init
Initialized empty Git repository in D:/learngit/.git/
```

初始化一个 Git 仓库，使用 git init 命令。Ls -ah 可视化添加文件到 Git 仓库，分两步：

```
[master (root-commit) cc6ac60] wrote a readme file
1 file changed, 2 insertions(+)
create mode 100644 readme.txt
```

1. 使用命令 `git add <file>`，注意，可反复多次使用，添加多个文件；
2. 使用命令 `git commit -m <message>`，完成。

注意：Windows 不要使用记事本，使用 notepad++

二、时光倒流

`git status` 命令可以让我们时刻掌握仓库当前的状态

`git diff` 查看 difference

1. 版本回退（修改指针 HEAD）

- 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`。

例子：`git reset --hard HEAD^` “返回上个版本”

`git reset --hard 1094a` “返回 id 开头为 1094a 的版本”

- 穿梭前，用 `git log` 可以查看提交历史，以便确定要回退到哪个版本。

`$ git log --pretty=oneline` 查看版本信息更简洁

- 要重返未来，用 `git reflog`。（记录历史命令）查看命令历史，以便确定要回到未来的哪个版本。

用 HEAD 表示当前版本，上一个版本就是 `HEAD^`，上上一个版本就是 `HEAD^^`，当然往上 100 个版本写成 `HEAD~100`。

2. 工作区和暂存区

- 工作区 (Working Directory):
在电脑里能看到的目录，比如 learngit 文件夹是一个工作区。
- 版本库 (Repository):

工作区有一个隐藏目录.git，这个不算工作区，而是 Git 的版本库。

Git 的版本库：名 stage（或者叫 index）的暂存区，git 自动创建的第一个分支 master，指向 master 的 HEAD 指针

`git status` 查看文件状态：

```
$ git status
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:   readme.txt

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

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

`git add` 命令实际上就是把要提交的所有修改放到暂存区（Stage），然后，执行 `git commit` 就可以一次性把暂存区的所有修改提交到分支。

3. 管理分支

Git 跟踪并管理的是修改，而非文件。

提交后，用 `git diff HEAD -- readme.txt` 命令可以查看工作区和版本库里面最新版本的差别

```
$ git diff HEAD -- readme.txt
diff --git a/readme.txt b/readme.txt
index e9d416c..9a8b341 100644
--- a/readme.txt
+++ b/readme.txt
@@ -1,4 +1,4 @@
 Git is a distributed version control system.
 Git is free software distributed under the GPL.
 Git has a mutable index called stage.
-Git tracks changes.
+Git tracks changes of files.
\ No newline at end of file
```

每次修改，如果不用 `git add` 到暂存区，那就不会加入到 `commit` 中。

4. 撤销修改

1. 只进行了修改

- `git checkout -- file` 可以丢弃工作区的修改，就是让这个文件回到最近一次 `git commit` 或 `git add` 时的状态
- `git restore <file>` 作用相同

2. 已经使用 git add 提交

`git checkout -- <file>`

3. 已经 git commit 提交了不合适的修改到版本库时，想要撤销本次提交，参考版本回退，不过前提是没有推送到远程库

5. 删除文件

命令 `git rm` 用于删除一个文件。如果一个文件已经被提交到版本库，那么你永远不用担心误删，但是要小心，你只能恢复文件到最新版本（`git checkout -- <file>`），你会丢失最近一次提交后你修改的内容。

三、 远程仓库

- Git 远程仓库，创建 ssh key:

打开 Shell（Windows 下打开 Git Bash），创建 SSH Key:

`ssh-keygen -t rsa -C 1799956902@qq.com`

`id_rsa` 是私钥，不能泄露出去，`id_rsa.pub` 是公钥

- Github 绑定

1. 添加远程库

```
$ git remote add origin git@github.com:Echhoo/learn-git.git
```

远程库名为 `origin`

要关联一个远程库，使用命令 `git remote add origin`

```
git@server-name:path/repo-name.git;
```

关联后，使用命令 `git push -u origin master` 第一次推送 master 分支的所有内容；

此后，每次本地提交后，只要有必要，就可以使用命令 `git push origin master` 推送最新修改；

2. 从远程库克隆

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 https，但 ssh 协议速度最快。

https: `https://github.com/Echhoo/gitskills.git`

ssh: `git@github.com:Echhoomichaeliao/gitskills.git`

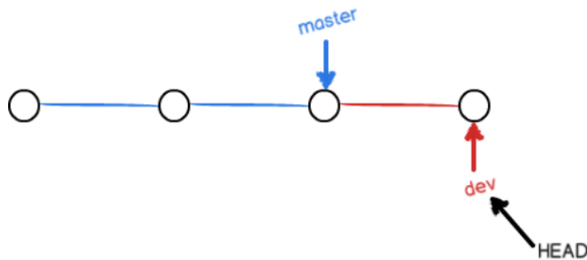
四、分支管理

Git 的分支无论创建、切换和删除分支，Git 在 1 秒钟之内就能完成。

1. 创建与合并分支

`HEAD` 严格来说不是指向提交，而是指向 `master`，`master` 才是指向提交的，所以，`HEAD` 指向的就是当前分支。`master` 主分支。

当我们创建新的分支，例如 `dev` 时，Git 新建了一个指针叫 `dev`，指向 `master` 相同的提交，再把 `HEAD` 指向 `dev`，就表示当前分支在 `dev` 上：



假如我们在 `dev` 上的工作完成了，就可以把 `dev` 合并到 `master` 上。再删除 `dev` 分支。

查看分支：`git branch` 当前分支前面会标一个 `*` 号。

创建分支：`git branch <name>`

切换分支：`git checkout <name>` 或者 `git switch <name>`

创建+切换分支：`git checkout -b <name>` 或者 `git switch -c <name>`

合并某分支到当前分支：`git merge <name>`

删除分支：`git branch -d <name>`

2. 解决冲突

当 Git 无法自动合并分支时，就必须首先解决冲突。解决冲突后，再提交，合并完成。

解决冲突就是把 Git 合并失败的文件手动编辑为我们希望的内容，再提交。

用 `git log --graph` 命令可以看到分支合并图。

```

$ git log --graph
* commit c0e6a0230b5de437338508fd6ca2209a46ba0575 (HEAD -> master)
  Merge: 9614891 9571363
  Author: echo <1799956902@qq.com>
  Date: Mon Jun 15 22:51:34 2020 +0800

    conflict

* commit 9571363d72d4f57c6028e30f41da015f5ceb488d (feature1)
  Author: echo <1799956902@qq.com>
  Date: Mon Jun 15 22:46:30 2020 +0800

    and bala

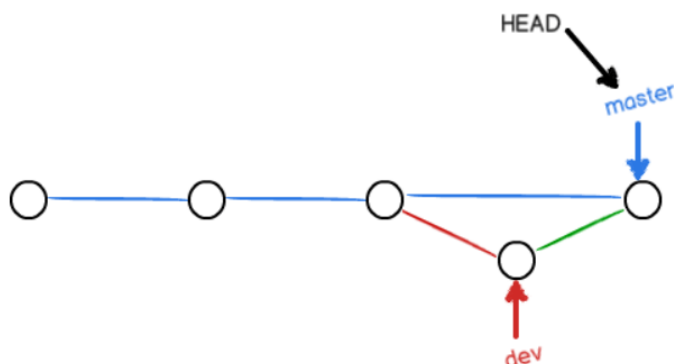
* commit 9614891899828c1558e5cd659a96f0a15a03de15
  Author: echo <1799956902@qq.com>
  Date: Mon Jun 15 22:47:16 2020 +0800

```

3. 分支管理策略

\$ git merge --no-ff -m "merge with no-ff" dev: 表示禁用 Fast forward

可以看到, 不使用 Fast forward 模式, merge后就像这样:



合并分支时, 加上 --no-ff 参数就可以用普通模式合并, 合并后的历史有分支, 能看出来曾经做过合并, 而 fast forward 合并就看不出曾经做过合并。

分支策略:

首先, master 分支应该是非常稳定的, 也就是仅用来发布新版本, 平时不能在上面干活; dev 分支是不稳定的, 到某个时候, 比如 1.0 版本发布时, 再把 dev 分支合并到 master 上, 在 master 分支发布 1.0 版本;

每个人都在 dev 分支上干活, 每个人都有自己的分支, 时不时地往 dev 分支上合并就可以了。

团队分支:



4. bug 分支

Git stash 隐藏工作区, 在 master 分支上修复, 就从 master 创建临时分支, 然后删除分支。接着回到 dev 分支, git status 查看 dev 工作区, 是干净的, 通过 git stash pop, 回到工作现场。

同样的 bug, 要在 dev 上修复, 我们只需要把 4c805e2 fix bug 101 这个提交所做的修改“复

制”到 dev 分支。注意：我们只想复制 `4c805e2 fix bug 101` 这个提交所做的修改，使用 `cherry-pick` 命令，避免重复劳动。

5. Feature 分支

开发一个新 feature，最好新建一个分支；

如果要丢弃一个没有被合并过的分支，可以通过 `git branch -D <name>` 强行删除。