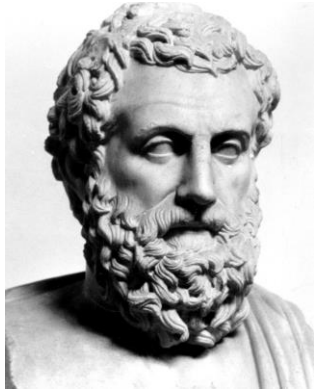
工程能力提升

Lesson-01



孟老师 & 开课吧人工智能学院课程组

2020.01

目录

1.1 简述

GIT兴起

- git是由Linux之父Linus开发的，在2005年4月3号开始开发，到 4.7 仅四天时间Git就可以投入使用了。到同年6月份，Linux已经在使用Git管理代码了。
- Linux版本管理最早是通过手工合并代码的方式进行的，不使用CVS的原因就是Linus坚决反对这种集中式的版本管理工具。而迫于其他开发者的压力，Linus最后决定使用BitKeeper一种分布式的版本管理工具。与集中式不同，使用分布式的版本管理工具每个人都可以在本地进行版本的管理，如日志提交、代码提交、创建tag和分支、合并分支等等操作。
- 2005年4月，Andrew Tridgell 为了开发一个可以与BitKeeper交互的工具，试图反编译BitKeeper。这让开发该软件的公司BitMover得知并取消了Linux社区免费试用BitKeeper的权利。这也成为了Linus开发Git的契机，促进了Git这一伟大作品的诞生。

1.1 简述

集中式（SVN） VS. 分布式（GIT）

- 集中式的版本控制系统都有一个单一的集中管理的服务器，保存所有文件的修订版本，而协同工作的人们都通过客户端连到这台服务器，取出最新的文件或者提交更新。
- 在分布式版本控制系统中，客户端并不只提取最新版本的文件快照，而是把原始的代码仓库完整地镜像下来。这么一来，任何一处协同工作用的服务器发生故障，事后都可以用任何一个镜像出来的本地仓库恢复。这类系统都可以指定和若干不同的远端代码仓库进行交互。籍此，你就可以在同一个项目中，分别和不同工作小组的人相互协作。你可以根据需要设定不同的协作流程。

SVN与GIT优缺点对比

- SVN优点： 1、 管理方便，逻辑明确，符合一般人思维习惯。 2、 易于管理，集中式服务器更能保证安全性。 3、 代码一致性非常高。 4、 适合开发人数不多的项目开发。
- SVN缺点： 1、 服务器压力太大，数据库容量暴增。2、 如果不能连接到服务器上，基本上不可以工作，看上面第二步，如果服务器不能连接上，就不能提交，还原，对比等等。 3、 不适合开源开发。
- GIT优点： 1、适合分布式开发，强调个体。 2、公共服务器压力和数据量都不会太大。 3、速度快、灵活。 4、任意两个开发者之间可以很容易的解决冲突。 5、离线工作
- GIT缺点： 1、学习周期相对而言比较长。 2、不符合常规思维。 3、代码保密性差，一旦开发者把整个库克隆下来就可以完全公开所有代码和版本信息。

思考题

快照 VS. 备份

1.2 安装配置

Windows安装

- 访问地址：https://git-scm.com/
- 点击下载：https://git-scm.com/download/gui/win
- 开始安装，下一步...

Linux安装

- Yum install git
- 下载源码：https://github.com/git/git.git 编译安装。

Mac安装

- 打开终端，执行命令brew install git@2.20.1

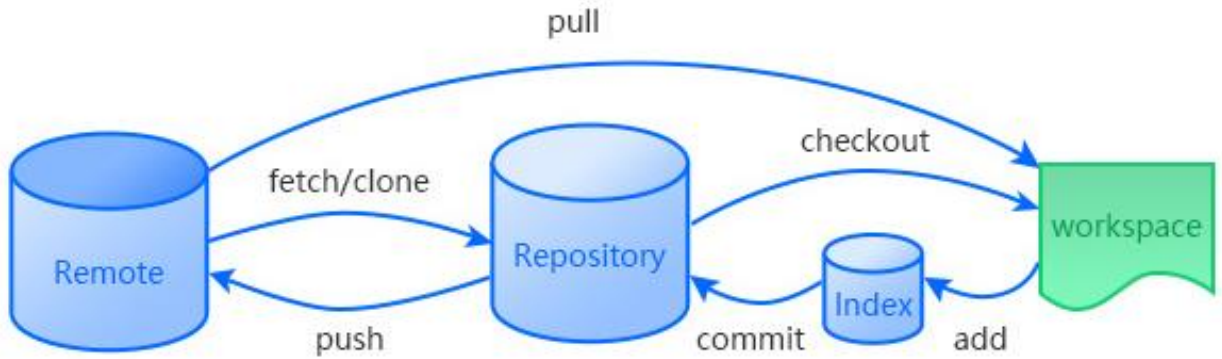
Git配置

- 配置用户名：git config --global user.name “xxx”
- 配置邮箱：git config --global user.email “xxx"
- 配置大小写敏感：git config --global core.ignorecase false
- 查看配置信息：git config --list

2.1 基本原理

Git工作区

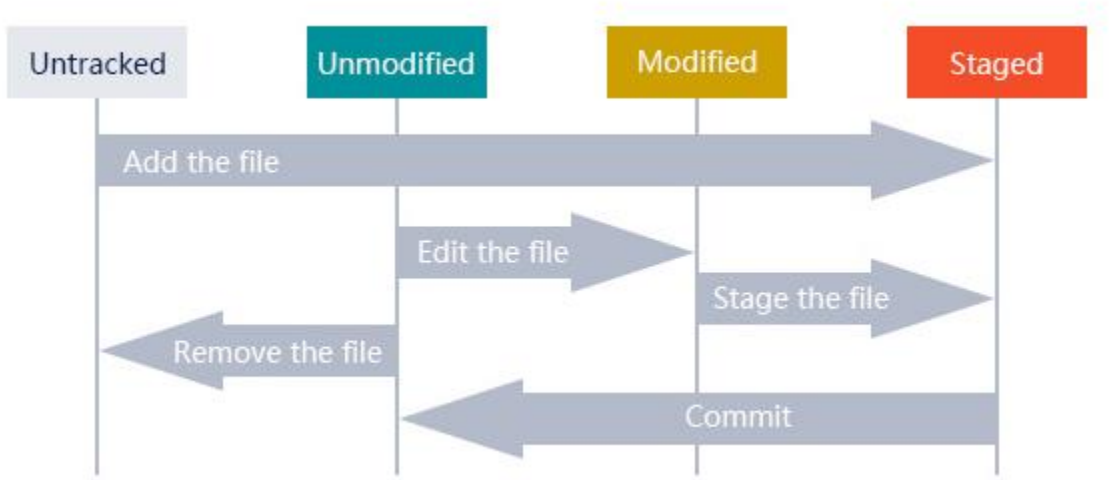
- Remote：远程仓库，托管代码的服务器，可以理解为是你项目组中的一台电脑用于远程数据交换（github）。
- Repository：仓库区（版本库），就是本地仓库，安全存放数据的位置，这里面有你提交所有版本的数据，其中HEAD指向最新放入的仓库版本。
- Index/Stage：暂存区，用于临时存放你的改动，事实上，它只是一个文件，保存即将提交到文件列表的信息。
- Workspace：工作区，就是你平时看到和编辑文件的区域。



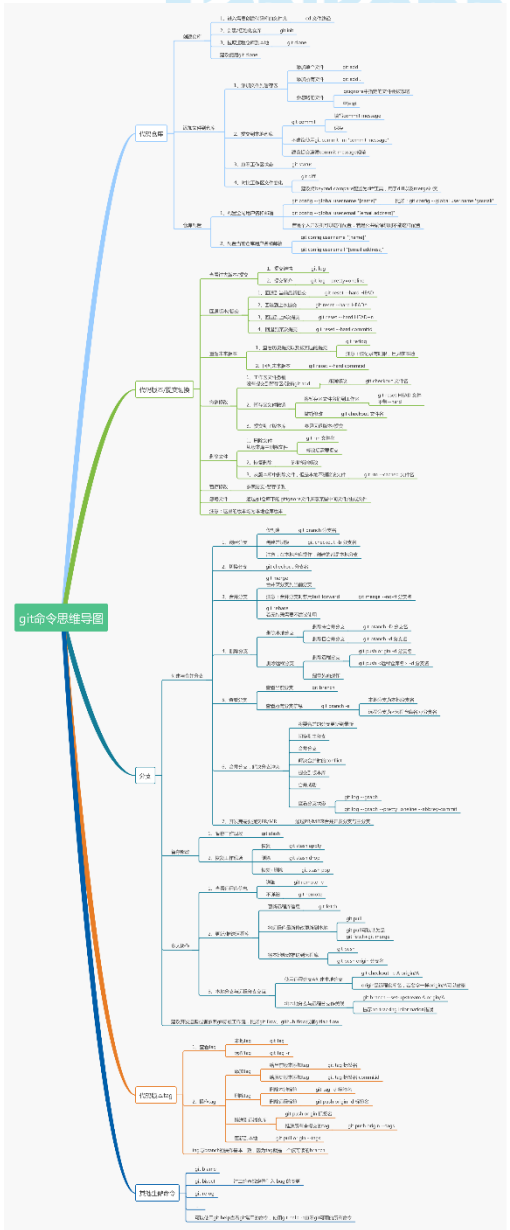
2.2 工作流程

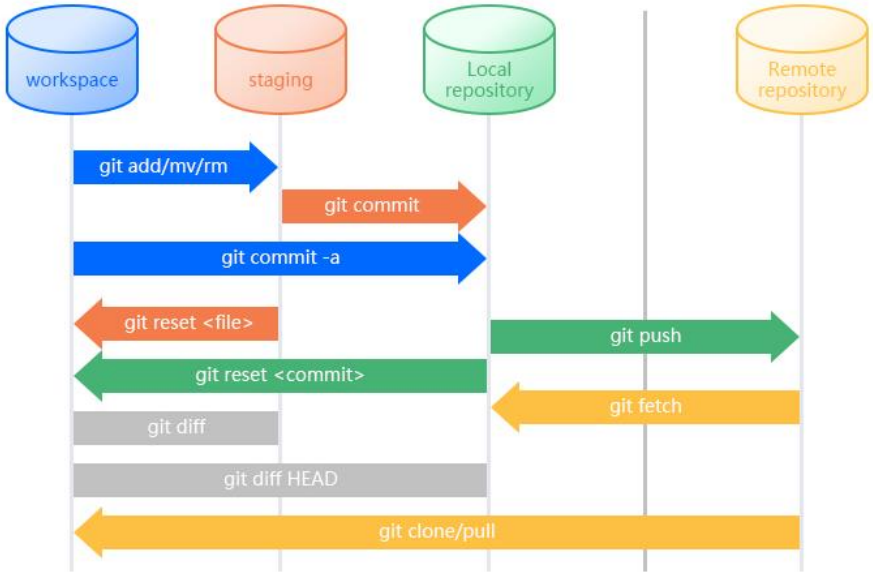
Git文件状态（版本控制就是对文件的版本控制）

- Untracked：未跟踪，在此文件夹中，但没有加入到git库，不参与版本控制，通过git add 状态变为staged。
- Unmodified：文件已经入库，未修改，即版本库中的文件快照内容与文件夹一致，这种类型的文件有两种去处，如果被修改，变为modified，如果被移除版本库git rm，则变为Untracked
- Modified：文件已修改，仅仅是修改，并没有进行其他操作，这个文件有两个去处，第一个是staged，第二个是unmodified。
- Staged：执行git commit 则将修改同步到库中，这时库中的文件和本地文件虽为一致，文件为Unmodified。执行git reset HEAD filename取消暂存，文件状态为modified。



3.1 基本操作





git add	添加文件	git mv	移动或重命名文件，目录或符号链接
git rm	从工作区和索引中删除文件	git commit	添加文件到本地仓库
git reset	回退版本/提交	git push	将本地修改推送到远程库
git fetch	更新远程库信息	git diff	显示提交和工作树等之间的更改
git clone	拉取远程仓库到本地	git pull	将远程库最新修改更新到本地
git status	显示工作目录和暂存区的状态		

3.1 基本操作

基本操作流程

- 选择一个合适的地方，创建一个空目录（版本库）

```
$ mkdir learngit
$ cd learngit
$ pwd
/data/git/learngit
```

- 初始化版本库

```
$ git init
$ ll -ah
drwxr-xr-x 7 root root 4.0K Jan 11 17:32 .git
```

- 新增文件

```
$ git add readme
$ git commit -m “add readme file
[master (root-commit) 3289742] add readme
 1 file changed, 2 insertions(+)
 create mode 100644 readme
```

提交了版本，发现有bug怎么办!!!

- 使用git reset –hard HEAD^ 回滚至上个版本或指定commit id。
- 使用git log查看日志

有一个问题

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

3.1 基本操作

基本操作流程

- 选择一个合适的地方，创建一个空目录（版本库）

```
$ mkdir learngit
$ cd learngit
$ pwd
/data/git/learngit
```

- 初始化版本库

```
$ git init
$ ll –ah
drwxr-xr-x 7 root root 4.0K Jan 11 17:32 .git
```

- 新增文件

```
$ git add readme
$ git commit –m “add readme file
[master (root-commit) 3289742] add readme
 1 file changed, 2 insertions(+)
 create mode 100644 readme
```

提交了版本，发现有bug怎么办!!!

- 使用git reset –hard HEAD^ 回滚至上个版本或指定commit id。
- 使用git log查看日志

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

3.1 基本操作

内容有错误，但是已经git add了怎么办!!!

- git reset HEAD readme，撤销暂存区的修改
- git checkout – readme，撤销工作区的修改

本地文件误删了怎么办，还能找回来吗!!!

- 别着急，可以继续使用git checkout – filename命令找回来文件
- 如果不是误删，就是不想要这个文件了，可以使用如下命令：

```
$ git rm filename
$ git commit –m “remove filename”
[master d46f35e] remove readme
 1 file changed, 1 deletion(-)
 delete mode 100644 readme
```

- 场景1：当你改乱了工作区某个文件的内容，想直接丢弃工作区的修改时，用命令git checkout -- file。
- 场景2：当你不但改乱了工作区某个文件的内容，还添加到了暂存区时，想丢弃修改，分两步，第一步用命令git reset HEAD <file>，就回到了场景1，第二步按场景1操作。
- 场景3：已经提交了不合适的修改到版本库时，想要撤销本次提交，参考版本回退一节，不过前提是没有推送到远程库。

4.1 远程仓库

设置github账号

- 注册github账号
- 设置SSH Keys，ssh-keygen -t rsa -C "youremail@exaple.com"
- 在github settings页面设置SSH Keys
- Github支持设置多个SSH Key。

设置远程仓库

- 在github新建名为learngit的新版本库
- 根据github的提示，在本地仓库运行命令：git remote add origin https://github.com/wonderclouds/learngit.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；

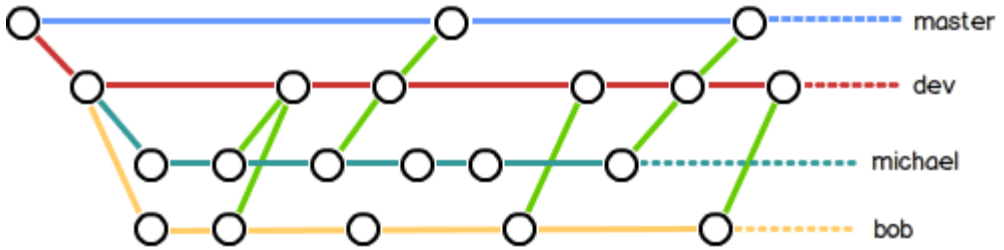
5.1 分支与标签管理

- 创建dev分支：git checkout –b dev /git switch -c dev
- 查看分支 git branch
- 分支内容提交：git commit –a –m “update file“
- 切换至master分支：git checkout master/git switch master
- 合并分支：git merge dev
- 删除dev分支：git branch –d dev

5.1 分支与标签管理

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；
- 那在哪干活呢？干活都在dev分支上，也就是说，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；
- 你和你的小伙伴们每个人都在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。
- 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合并就看不出来曾经做过合并。



5.1 分支与标签管理

- 创建标签：git tag v1.0
- 查看标签：git tag
- 创建带有描述信息的标签：git tag -a v0.1 -m "version 0.1 released" 1094adb
- 用命令git show <tagname>可以看到说明文字
- 如果打错了，可以删除：git tag –d v0.1
- 还可以将标签推到远程仓库：git push origin v1.0
- 删除远程标签需要先删除本地标签：git tag -d v0.9/git push origin :refs/tags/v0.9

6.1 GITHUB使用及个人主页搭建

QA