git的初步认识

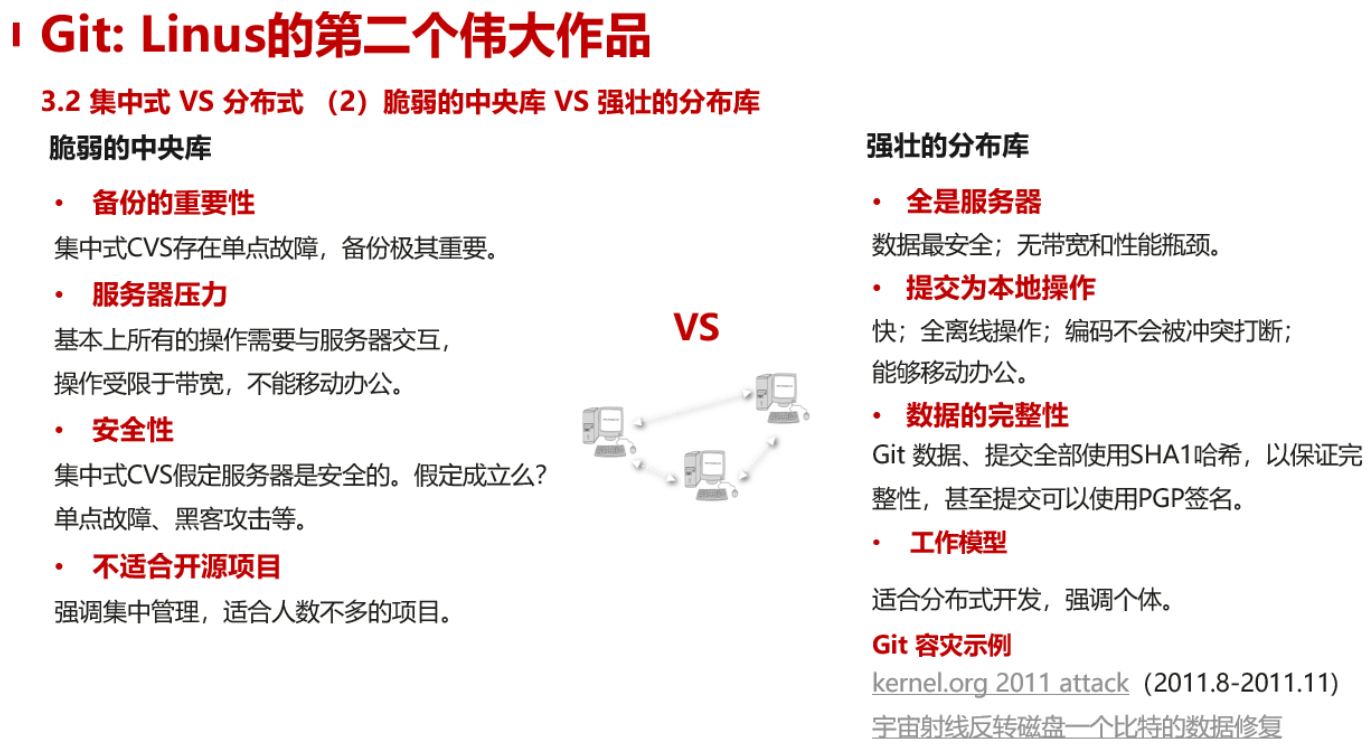
什么是版本控制 有哪些

版本控制是一种记录一个或若干文件内容变化，以便将来查阅特定版本修订情况的系统，方便查看更改历史，备份以及恢复以前的版本，保证多人的协作不出问题。早期的版本有diff patch
RCS(最早期的本地版本控制） CVS(集中式） SVN(集中式） GIT(分布式)

集中式与分布式的区别

集中式版本控制工具，几乎所有的动作都需要服务器参与，并且数据安全性与服务器关系很大。

Git是分布式版本控制工具，除了与服务器之前进行按需同步之外，所有的提交操作都不需要服务器。



git 的安装

这里我就不用详细说了，网上资源很多 ，这里推荐一个链接很详细，里面码云和git的配置，GitHub也是同理

https://www.it235.com/%E5%AE%9E%E7%94%A8%E5%B7%A5%E5%85%B7/Git/git.html#git%E5%AE%89%E8%A3%85

git的基本命令

在了解git命令之前需要了解git的三种工作区域和文件的状态

工作区

1.版本库(Repository)

在工作区中有一个隐藏目录`.git`，这个文件夹就是`Git`的版本库，里面存放了`Git`用来管理该工程的所有版本数据，说白了就叫
本地仓库

2.工作区(Working Directory)

日常工作的代码文件或者文档所在的文件夹

3.暂存区(stage)

一般存放在工程根目录 `.git/index`文件中，所以我们也可以把暂存区叫作索引（`index`）

文件的状态

1.已提交(committed)

就是说该文件已经被安全地保存在本地数据库中了

2.已暂存(staged)

就是把已修改的文件放在下次提交时要保存的清单中。已暂存`(staged)`的文件就是被已跟踪`(tracked)`的文件，是指该文件被
纳入了版本控制管理的文件

`git add [file]` `file` 是你需要纳入版本控制文件的文件名

如果需要添加多个文件到暂存区

```
git add [file1] [file2] ...
```

如果是当前目录下所有的文件使用

```
git add .
```

3.已修该(modified)

```
修改了某个已跟踪的文件，但还没有提交保存,这个时候你就需要把你修改的文件提交到本地仓库中
使用命令如下
git add file    file 你要保存的文件名
git commit -m "message"  message 就是你要提交的信息
也可以使用
git commit -am  "message"  我建议使用这一条命令,可以直接提交以被跟踪的文件到本地仓库
```

常用的命令

下面我将列出常用的git命令，在这里尤其需要提示一下 git status 和git log这两个命令，这两个命令一定要常用，很多时候我们合并更新分支出错的时候，可以根据这两个命令看当前的一个状态和解决方法提示

```
工程准备 git init/git clone

新增/删除/移动文件到暂存区 git add/ git rm/ git mv

查看工作区 git diff/ git status

提交更改的文件 git commit

查看日志 git log

推送远端仓库 git push

分支管理 git branch/git checkout /git branch –d/git pull

分支合并 git merge/git rebase
```

本地基本提交推送

```
$ git clone https://github.com/Bai794/git_dem2.git
Cloning into 'git_dem2'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 9 (delta 1), reused 6 (delta 1), pack-reused 0
Receiving objects: 100% (9/9), done.
Resolving deltas: 100% (1/1), done.

bai@BAI MINGW64 /d/git/demo2
$ cd git_dem2/

bai@BAI MINGW64 /d/git/demo2/git_dem2 (main)
$ ls
README.md  xioabai.cpp

bai@BAI MINGW64 /d/git/demo2/git_dem2 (main)
$ touch hideme.cpp

bai@BAI MINGW64 /d/git/demo2/git_dem2 (main)
$ ls
README.md  hideme.cpp  xioabai.cpp

bai@BAI MINGW64 /d/git/demo2/git_dem2 (main)
$ git add .

bai@BAI MINGW64 /d/git/demo2/git_dem2 (main)
$ git commit -am "fisrst"
[main f0fa9ed] fisrst
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 hideme.cpp

bai@BAI MINGW64 /d/git/demo2/git_dem2 (main)
$ git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 302 bytes | 302.00 KiB/s, done.
Total 3 (delta 0), reused 1 (delta 0), pack-reused 0
To https://github.com/Bai794/git_dem2.git
   1975047..f0fa9ed  main -> main
```

首先我先利用 git clone 克隆远端服务器的一个工程 在新建了一个 hideme.cpp文件 在推送到远端

本地分支的新建并推送到远端

很多时候我们克隆下来一个master或者main主分支，会新建一个自己的分子修改代码并提交，以免影响到主分子

```
bai@BAI MINGW64 /d/git/demo2/git_dem2 (main)
$ git checkout -b hideme main
Switched to a new branch 'hideme'

bai@BAI MINGW64 /d/git/demo2/git_dem2 (hideme)
$ ls
README.md  hideme.cpp  xioabai.cpp

bai@BAI MINGW64 /d/git/demo2/git_dem2 (hideme)
$ git rm xioabai.cpp
rm 'xioabai.cpp'

bai@BAI MINGW64 /d/git/demo2/git_dem2 (hideme)
$ ls
README.md  hideme.cpp
```

```
bai@BAI MINGW64 /d/git/demo2/git_dem2 (hideme)
$ ls
README.md  hideme.cpp

bai@BAI MINGW64 /d/git/demo2/git_dem2 (hideme)
$ git commit -am "rm xiaobai.cpp"
[hideme dfc66c4] rm xiaobai.cpp
 1 file changed, 1 deletion(-)
 delete mode 100644 xioabai.cpp

bai@BAI MINGW64 /d/git/demo2/git_dem2 (hideme)
$ git push origin hideme
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 212 bytes | 212.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Bai794/git_dem2.git
   f0fa9ed..dfc66c4  hideme -> hideme
```

通过 `git checkout -b new_branch old_branch` 新建了一个hideme分支并把当前分支目录切换到hideme,后面跟的一个main 表示复制一下main
分支到我新建的分支下

然后我在hideme分支下删除了一个xiaobai.cpp文件并推送到远端hideme 分支

回退版本

现在我后悔删除 xiaobai.cpp文件 怎么回退版本，恢复xiaobai.cpp

首先我们可以通过 git log -2 打印最近两次版本的操作 然后通过diff 来比较两个版本的差异

```
$ git log -2
commit dfc66c4ea5adb7d44f997c7659c87624916d6f93 (HEAD -> hideme, origin/hideme)
Author: xiaobai <1269724114@qq.com>
Date:   Sun Dec 19 14:40:13 2021 +0800

    rm xiaobai.cpp

commit f0fa9edc439640767a48588af4df862751dd1a84 (origin/main, origin/HEAD, main)
Author: xiaobai <1269724114@qq.com>
Date:   Sun Dec 19 14:04:13 2021 +0800

    fisrst

bai@BAI MINGW64 /d/git/demo2/git_dem2 (hideme)
$ git diff ^C

bai@BAI MINGW64 /d/git/demo2/git_dem2 (hideme)
$ git diff dfc66c4ea f0fa9edc4
diff --git a/xioabai.cpp b/xioabai.cpp
new file mode 100644
index 0000000..861ec69
--- /dev/null
+++ b/xioabai.cpp
@@ -0,0 +1 @@
+xiaobai is handsome
```

我们可以通过git reset --hard +版本号实现版本回退

也可以 git reset --hard HEAD^ 回退上一次版本

git reset --hard HEAD~3 回到之前三次的版本 依次类推

```
bai@BAI MINGW64 /d/git/demo2/git_dem2 (hideme)
$ git reset --hard 19750473437c563333d5c8b6cc085612f1ccb5f8
HEAD is now at 1975047 second

bai@BAI MINGW64 /d/git/demo2/git_dem2 (hideme)
$ ls
README.md  xioabai.cpp

bai@BAI MINGW64 /d/git/demo2/git_dem2 (hideme)
$ ls
README.md  xioabai.cpp

bai@BAI MINGW64 /d/git/demo2/git_dem2 (hideme)
$ cat xioabai.cpp
xiaobai is handsome
```

好，现在我们本地的xioabai.cpp文件已经恢复了 ， 现在我们想更新到远程仓 怎么办

如果我们使用 `git push origin <branch name>`(分支名称) 此时会报错,这是因为我们使用 `git reset` 是强制其回到某个节点，那个节点之后的

节点就不存在了,而远端的仓库却保存着之后的节点，所以会报出你的仓库版本太低的错误而不能推送到远端

这个时候我们只有强制推送，采用如下命令

`git push -f origin <branch name>`

所以采用git reset 会删除节点之后的节点，这种方式不推荐

我们可以采用 git revert -n 命令来避免以上问题，具体步骤如下

1.使用`git log` 找到你误提交之前的版本号 `git log --name-status` 看各个节点修改的详细信息

2.`git revert -n` 版本号

3.`git commit -m xxxx` 提交

4.`git push` 推送到远程

```
bai@BAI MINGW64 /d/git/demo2/git_dem2 (hideme)
$ git revert -n 19750473437c563333d5c8b6cc085612f1ccb5f8

bai@BAI MINGW64 /d/git/demo2/git_dem2 (hideme|REVERTING)
$ git commit  -m "second"
[hideme 8070f97] second
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 xiaobai.cpp

bai@BAI MINGW64 /d/git/demo2/git_dem2 (hideme)
$ git push origin hideme
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 449 bytes | 224.00 KiB/s, done.
Total 4 (delta 0), reused 1 (delta 0), pack-reused 0
To https://github.com/Bai794/git_dem2.git
   1c723c0..8070f97  hideme -> hideme

bai@BAI MINGW64 /d/git/demo2/git_dem2 (hideme)
$
```

大家可以参考链接 https://www.cnblogs.com/aligege/p/10221174.html 详细看一下git reset和git revert的区别

恢复新版本

如果有一天你又想恢复到新版本怎么办 git log 会找不到新版本的commit id，这时候可以使用 git reflog 命令

git reflog 可以查看所有分支的所有操作记录（包括commit和reset的操作），包括已经被删除的commit记录，git log则不能察看已经删除了的commit记录，而且跟进结果可以回退道某一个修改

```
bai@BAI MINGW64 /d/git/demo2/git_dem2 (hideme)
$ git reflog
f0fa9ed (HEAD -> hideme, origin/main, origin/HEAD, main) HEAD@{0}: reset: moving to HEAD^
c7ca43e (origin/hideme) HEAD@{1}: pull origin hideme: Fast-forward
f0fa9ed (HEAD -> hideme, origin/main, origin/HEAD, main) HEAD@{2}: merge origin: Fast-forward
1975047 HEAD@{3}: reset: moving to 19750473437c563333d5c8b6cc085612f1ccb5f8
c7ca43e (origin/hideme) HEAD@{4}: commit: reback
f0fa9ed (HEAD -> hideme, origin/main, origin/HEAD, main) HEAD@{5}: reset: moving to f0fa9edc439640767a48588af4df862751dd1a
dfc66c4 HEAD@{6}: commit: rm xiaobai.cpp
f0fa9ed (HEAD -> hideme, origin/main, origin/HEAD, main) HEAD@{7}: checkout: moving from main to hideme
f0fa9ed (HEAD -> hideme, origin/main, origin/HEAD, main) HEAD@{8}: checkout: moving from hideme to main
f0fa9ed (HEAD -> hideme, origin/main, origin/HEAD, main) HEAD@{9}: checkout: moving from main to hideme
f0fa9ed (HEAD -> hideme, origin/main, origin/HEAD, main) HEAD@{10}: commit: fisrst
1975047 HEAD@{11}: clone: from https://github.com/Bai794/git_dem2.git

bai@BAI MINGW64 /d/git/demo2/git_dem2 (hideme)
$ git reset --hard c7ca43e
HEAD is now at c7ca43e reback

bai@BAI MINGW64 /d/git/demo2/git_dem2 (hideme)
$ ls
README.md  hideme.cpp
```

可以看到 xioabai.cpp又没有了

分支的合并

我们发现主分支main代码有点bug，需要你及时修改，我们可以新建一个分支copy 一下 main分支，在我们新建的分支修改bug完后可以通过git merge 合并到main分支

```
bai@BAI MINGW64 /d/git/demo2/git_dem2 (main)
$ ls
README.md  xiaobai.cpp

bai@BAI MINGW64 /d/git/demo2/git_dem2 (main)
$ git checkout -b slove main
Switched to a new branch 'slove'

bai@BAI MINGW64 /d/git/demo2/git_dem2 (slove)
$ ls
README.md  xiaobai.cpp

bai@BAI MINGW64 /d/git/demo2/git_dem2 (slove)
$ cat xiaobai.cpp
dhfuihdi
dchiudhciu
dchihci


bai@BAI MINGW64 /d/git/demo2/git_dem2 (slove)
$ echo "fjjhifehihfi">>xiaobai.cpp

bai@BAI MINGW64 /d/git/demo2/git_dem2 (slove)
$ cat xiaobai.cpp
dhfuihdi
dchiudhciu
dchihci

fjjhifehihfi
```

在我们新建分支slove 添加了一句话，在slove分支上提交后就可以在切换到main分支使用git merge 合并到main分支

```
bai@BAI MINGW64 /d/git/demo2/git_dem2 (slove)
$ git commit -am "slove"
warning: LF will be replaced by CRLF in xiaobai.cpp.
The file will have its original line endings in your working directory
[slove 23feef6] slove
 1 file changed, 1 insertion(+)

bai@BAI MINGW64 /d/git/demo2/git_dem2 (slove)
$ ls
README.md  xiaobai.cpp

bai@BAI MINGW64 /d/git/demo2/git_dem2 (slove)
$ git checkout  main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

bai@BAI MINGW64 /d/git/demo2/git_dem2 (main)
$ git merge slove
Updating 1bb382b..23feef6
Fast-forward
 xiaobai.cpp | 1 +
 1 file changed, 1 insertion(+)

bai@BAI MINGW64 /d/git/demo2/git_dem2 (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

bai@BAI MINGW64 /d/git/demo2/git_dem2 (main)
$ ls
README.md  xiaobai.cpp

bai@BAI MINGW64 /d/git/demo2/git_dem2 (main)
$ cat xiaobai.cpp
dhfuihdi
dchiudhciu
dchihci

fjjhifehihfi
```

这部分可以参考git 官网的链接https://git-scm.com/book/zh/v2/Git-%E5%88%86%E6%94%AF-%E5%88%86%E6%94%AF%E7%9A%84%E6%96%B0%E5%BB%BA%E4%B8%8E%E5%90%88%E5%B9%B6