《开源软件设计与开发》课程总结

姓名：钟善毫　　学号：51194507024

1.开源理解

Git 简史

同生活中的许多伟大事件一样，Git 诞生于一个极富纷争大举创新的年代。Linux 内核开源项目有着为数众广的参与者。绝大多数的 Linux 内核维护工作都花在了提交补丁和保存归档的繁琐事务上（1991－2002 年间）。到 2002 年，整个项目组开始启用分布式版本控制系统 BitKeeper 来管理和维护代码。

到了 2005 年，开发 BitKeeper 的商业公司同 Linux 内核开源社区的合作关系结束，他们收回了免费使用 BitKeeper 的权力。这就迫使 Linux 开源社区（特别是 Linux 的缔造者 Linus Torvalds ）不得不吸取教训，只有开发一套属于自己的版本控制系统才不至于重蹈覆辙。他们对新的系统制订了若干目标：

速度

简单的设计

对非线性开发模式的强力支持（允许上千个并行开发的分支）

完全分布式

有能力高效管理类似 Linux 内核一样的超大规模项目（速度和数据量）

自诞生于 2005 年以来，Git 日臻成熟完善，在高度易用的同时，仍然保留着初期设定的目标。它的速度飞快，极其适合管理大项目，它还有着令人难以置信的非线性分支管理系统（见第三章），可以应付各种复杂的项目开发需求。

对 Git 工具的理解

Git 是用于 Linux 内核开发的版本控制工具。与常用的版本控制工具 CVS, Subversion 等不同,它采用了分布式版本库的方式,不必服务器端软件支持,使源代码的发布和交流极其方便。Git 的速度很快，这对于诸如 Linux kernel 这样的大项目来说自然很重要。 Git 最为出色的是它的合并跟踪（merge tracing）能力。

Git 实践

如何创建自己的 git 仓库？

（1）创建 SSH Key

```
$ ssh-keygen -t rsa -C "youremail@example.com"
```

在.ssh 目录下有 id_rsa 和 id_rsa.pub 两个文件。

（2）在 github 上将 id_rsa.pub 公钥添加到 SSH Keys 中

（3）在本地上创建一个 git 仓库

```
$ git init
```

（4）关联远程库

```
$ git remote add origin git@github.com:zhongshanhao/learngit.git
```

origin 是远程仓库的名字，关联远程库后就可以在本地管理 git 仓库了，这里的创建的仓库是 learngit。

如何向开源社区贡献自己的代码？

（1）在 github 上 fork 项目源代码到自己的远程仓库

（2）在本地 clone 仓库

```
git clone git@github.com:zhongshanhao/tidb.git
```

（3）添加远程仓库分支 upstream，该分支是项目源代码所在的地方，不是自己克隆的远程仓库

```
git remote add upstream https://github.com/pingcap/tidb
```

（4）在开发之前先同步远程仓库

```
git fetch upstream
git merge upstream/master

#  或者
git pull upstream master
```

（5）然后在本地仓库创建分支进行开发

```
git checkout -b my_branch
```

（6）完成代码开发后，将分支推送到远程仓库

```
git push origin my_branch:my_branch
```

（7）在 github 上选择 my_branch 分支，点击 new pull request 创建新的 PR

（8）等待 pr 完成，可以删除该分支

```
git branch -d my_branch
```

命令拓展

（1）暂存工作现场

```
git log upstream/master    # 查看 upstream/master 分支最新的 commit

# 当当前工作没有提交到本地仓库时，可以将工作现场存储起来
git stash
git stash list # 查看暂存区列表
git stash pop # 恢复工作现场，删除暂存区内容
git stash apply stash@{0}    # 恢复指定暂存区，且不删除该暂存区内容
git stash drop # 删除暂存区内容
```

（2）当 merge 出现冲突

```
git status #定位冲突文件
#修改文件解决冲突
git add .
git commit -m "solve conflict"
```

rebase 命令

（1）变基操作，将 my_branch 分支变基，即将该分支的分叉出改为最新的 master 头结点处

```
git checkout my_branch
git rebase master
git push origin my_branch:my_branch
```

（2）变基出现冲突时

```
git rebase master #出现冲突
#解决冲突
git add .
git rebase --continue #继续变基
```

（3）git push 出现冲突时

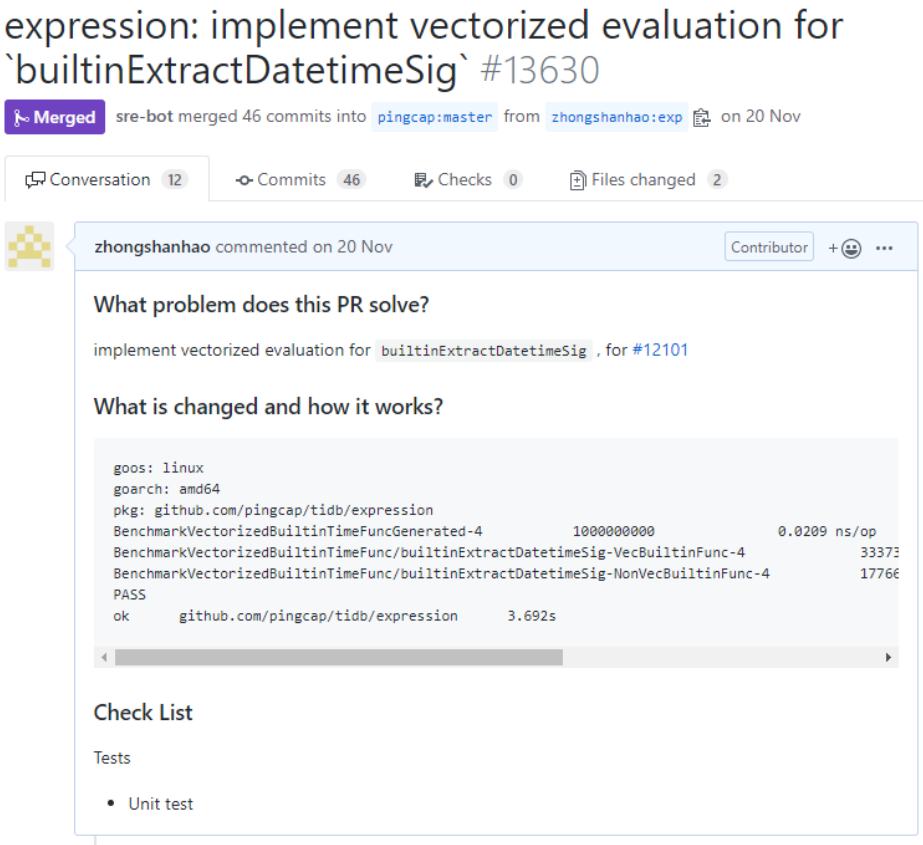
```
git fetch origin    #拉取远程分支
git merge origin/my_branch    #合并对应分支，合并冲突在另外解决

#若有冲突,定位文件,修改文件,提交修改
git status
git add .
git commit -m "fix conflict"

#最后 push
git push origin my_branch:my_branch
```

2、开源贡献

我参与了开源项目 TiDB，在该开源项目中，我完成了在计算框架下实现 builtinExtractDatetimeSig 函数的向量化计算 ，下图是我在 TiDB 提交的 PR：



具体完成的工作和代码贡献在此链接： https://github.com/pingcap/tidb/pull/13630 。该 PR 已经被 Merged。

具体完成工作（代码贡献）：

所需要向量化的 builtinExtractDatetimeSig 函数：

```
func (b *builtinExtractDatetimeSig) vecEvalInt(input *chunk.Chunk, result *chunk.Column) error {
-   return errors.Errorf("not implemented")
+   n := input.NumRows()
+   buf, err := b.bufAllocator.get(types.ETString, n)
+   if err != nil {
+       return err
+   }
+   defer b.bufAllocator.put(buf)
+   if err := b.args[0].VecEvalString(b.ctx, input, buf); err != nil {
+       return err
+   }
+
+   buf1, err := b.bufAllocator.get(types.ETDatetime, n)
+   if err != nil {
+       return err
+   }
+   defer b.bufAllocator.put(buf1)
+   if err := b.args[1].VecEvalTime(b.ctx, input, buf1); err != nil {
+       return err
+   }
+
+   result.ResizeInt64(n, false)
+   i64s := result.Int64s()
+   ds := buf1.Times()
+   result.MergeNulls(buf, buf1)
+   for i := 0; i < n; i++ {
+       if result.IsNull(i) {
+           continue
+       }
+       res, err := types.ExtractDatetimeNum(&ds[i], buf.GetString(i))
+       if err != nil {
+           return err
+       }
+       i64s[i] = res
+   }
+   return nil
}
```

添加测试框架：

添加的数据生成器：

```
+ type dateTimeUnitStrGener struct{}
+
+ func (g *dateTimeUnitStrGener) gen() interface{} {
+   dateTimes := []string{
+       "DAY",
+       "WEEK",
+       "MONTH",
+       "QUARTER",
+       "YEAR",
+       "DAY_MICROSECOND",
+       "DAY_SECOND",
+       "DAY_MINUTE",
+       "DAY_HOUR",
+       "YEAR_MONTH",
+   }
+
+   n := rand.Int() % len(dateTimes)
+   return dateTimes[n]
+ }
```

```
ast.Extract: {
    {retEvalType: types.ETInt, childrenTypes: []types.EvalType{types.ETString, types.ETDatetime}, geners: []dataGenerator{&dateTimeUnitStrGener{},
},
```

最后的成果：

What problem does this PR solve?

implement vectorized evaluation for `builtinExtractDatetimeSig` , for #12101

What is changed and how it works?

```
goos: linux
goarch: amd64
pkg: github.com/pingcap/tidb/expression
BenchmarkVectorizedBuiltinTimeFuncGenerated-4           1000000000               0.0209 ns/op
BenchmarkVectorizedBuiltinTimeFunc/builtinExtractDatetimeSig-VecBuiltinFunc-4              33373
BenchmarkVectorizedBuiltinTimeFunc/builtinExtractDatetimeSig-NonVecBuiltinFunc-4           17766
PASS
ok      github.com/pingcap/tidb/expression      3.692s
```

Check List

Tests

- Unit test

3、课程反馈

本次的开源课程中，我学习到了很多关于开源相关的东西，也了解到了很多强大的开源项目，见识到了很多做开源工作的大牛，同时我也希望该课程能够越来越好，以下是我对该课程我提的一点意见。

希望课程能够增加 Git 实践的相关内容，这样能够让同学们熟练使用 Git 的同时将时间更加集中于代码的贡献上，而不是在 Git 的使用上。

4、参考文献

[1]Pro Git 中文版. https://gitee.com/progit/index.html