

组会

薄纪铮

2023 年 10 月 10 日

目录

1 在 makefile 中执行 shell 命令 (git bash)

2 git 的使用

3 THO 计算相移

1 在 makefile 中执行 shell 命令 (git bash)

首先定义 shell 命令对应的变量:

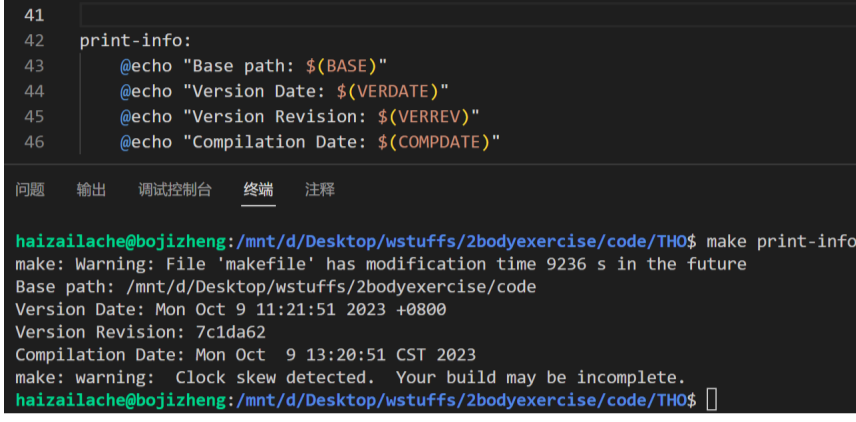
```
#定义一些 git bash 的 shell 命令变量
BASE := $(shell expr $(CURDIR) : "\(.*\)/.*")
VERDATE := $(shell git log -1 --format=%cd)
VERREV := $(shell git log -1 --pretty=format:"%h")
COMPDATE := $(shell date)
```

然后在 makefile 的后面部分调用这些变量:

```
.PHONY: print-info
print-info:
    @echo "Base path: $(BASE)"
    @echo "Version Date: $(VERDATE)"
    @echo "Version Revision: $(VERREV)"
    @echo "Compilation Date: $(COMPDATE)"
```

注意: @echo 是 makefile 中的一个命令, 相当于 print 到终端的意思, 在 makefile 中涉及到命令的行都需要以 tab 制表符开头, 不能使用空格。

然后可以在终端使用命令 make print-info 来执行这些 shell 命令以打印出一些源文件的信息。



```
40 .PHONY: print-info
41
42 print-info:
43     @echo "Base path: $(BASE)"
44     @echo "Version Date: $(VERDATE)"
45     @echo "Version Revision: $(VERREV)"
46     @echo "Compilation Date: $(COMPDATE)"
问题 输出 调试控制台 终端 注释
haizailache@bojizheng: /mnt/d/Desktop/wstuffs/2bodyexercise/code/THO$ make print-info
make: Warning: File 'makefile' has modification time 9236 s in the future
Base path: /mnt/d/Desktop/wstuffs/2bodyexercise/code
Version Date: Mon Oct 9 11:21:51 2023 +0800
Version Revision: 7c1da62
Compilation Date: Mon Oct 9 13:20:51 CST 2023
make: warning: Clock skew detected. Your build may be incomplete.
haizailache@bojizheng: /mnt/d/Desktop/wstuffs/2bodyexercise/code/THO$
```

图 1: make print-info 执行的示例结果

2 git 的使用

GIT 的概念: 'Git is open source and free source control management or what is referred to as SCM', 翻译一下大致就是一个开源管理器。

以 git bash 终端为例, 首先在终端设置全局的用户设置, 包括用户名邮箱等, 我们使用命令

```
git config --global user.name "yourname"
git config --global user.email xxxxxxxx
```

然后需要用 git 创建一个仓库来做源码的追踪 track 管理, 因此我们使用命令:

```
git init
```

或者 git initialize 也行, 这样就在当前目录下创建了一个 .git 的隐藏文件夹

然后查看 git 的一些作用在当前 branch 以及一些源文件上的常用命令:

首先是

```
git status
```

该命令告诉我们: 当前所处的 branch 分支的层级, 以及当前 branch 里面各源文件的状态 (tracked/untracked)

```
On branch master
```

```
No commits yet
```

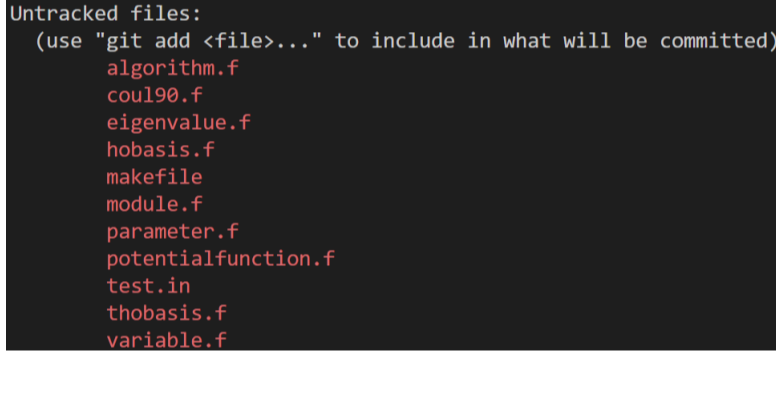
```
Untracked files:
```

```
(use "git add <file> ..." to include in what will be committed)
```

然后将 untracked 的文件去 track 一下, 使用 git add 命令, 如

```
git add THO.f
```

然后 git status 一下会发现能 commit 的文件是 THO.f, 剩下来的都是 untracked 的文件



```
Changes to be committed:
(use "git rm --cached <file> ..." to unstage)
    new file:   THO.f

Untracked files:
(use "git add <file> ..." to include in what will be committed)
    algorithm.f
    coul90.f
    eigenvalue.f
    hobasis.f
    makefile
    module.f
    parameter.f
    potentialfunction.f
    test.in
    thobasis.f
    variable.f
```

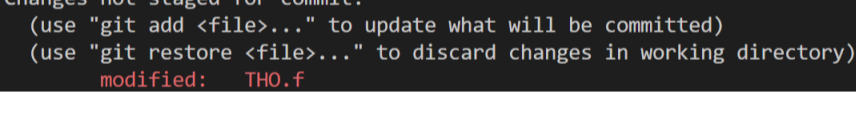
要 track 所有的文件, 需使用命令

```
git add --all
git add -A
git add .
```

上述三个命令都可以使用。

在对所有文件做了 track 之后, 我们就可以对源文件做修改

然后 git status 查看源文件状态




```
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:   THO.f
```

图 3: 修改之后可以看到有的文件是 modified 状态, 意思是受到了调整

然后我们可以查看具体修改的内容, 使用命令

```
git diff
```



```
$ git diff
diff --git a/THO.f b/THO.f
index e872212..73ced37 100644
--- a/THO.f
+++ b/THO.f
@@ -55,7 @@ ccccccc
ccccccc
-      mu=mu*(mass_2*mass_1)/(mass_1+mass_2) !约化质量
+      alpha=100/200/b**2/sqrt(mu*omega/hbar) !HOBASIS的无量纲参数alpha=mu*omega/hbar
+      alpha=100/200/b**2/sqrt(mu*omega/hbar) !HOBASIS的无量纲参数alpha=mu*omega/hbar
+      11/b**2-mu*omega/hbar,而basis的标准形式下的系数是mu*omega/2/hbar
write(*,*) 'omega=',hbar/mu/b**2
write(*,*) 'nbasis=',n_basis
```

图 4: git diff 查看具体修改的内容

在做完修改之后, 我们需要提交一个当前的修订完的版本, 使用命令

```
git commit -a -m 'messages'
```

该命令是提交当前目录的所有文件到仓库

'messages' 部分是一个必填项, 指的是对该版本的描述

对一个项目的多次修改的历史可以通过日志查看, 对应命令

```
git log
```

git 的另一个重要功能是 branch 分支功能, 首先我们建立一个分支

```
git branch newbranchname
```

使用上述命令创建了一些分支之后, 我们可以用 git branch 命令查询有哪些分支, 得到:

```
$ git branch
* master
newbranchname
```

其中星号 * 表示当前所处 branch, 我们可以用 git switch newbranchname 命令来切换到别的分支。在做完 branch 的事情之后, 可以将它们合并到 main branch 里面:

具体使用

```
git merge -m 'messages' branchname
```

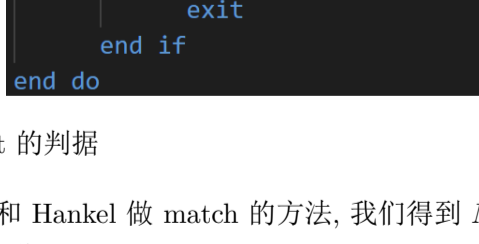
然后可以删除这个 branch(因为已经 merge 了), 使用

```
git branch -d branchname
```

因此简单总结一下 git 的工作流:

- 建立一些 branch
- 在 branch 里做完事情并 commit
- 把这个 branch 去 merge 到 main branch 里面
- delete 这个 branch

3 THO 计算相移



```
do i=1,n_int
    if(abs(vpot(i))<3e-3) then
        n_match=i
        exit
    end if
end do
```

图 5: match point 的判据

效仿 numerov 里和 Hankel 做 match 的方法, 我们得到 $N_{basis} = 15$ 时各个正特征能量下的相移

E(MeV)	$S_0(\text{THO})$	$S_0(\text{Modified Numerov})$
0.10009	(0.86214,-0.50666)	(0.86709,-0.49814)
0.40888	(0.55203,-0.83382)	(0.52447,-0.85143)
0.95525	(7.35425E-002,-0.99729)	(9.26577E-002,-0.99570)
1.79413	(-0.27095,-0.96259)	(-0.32030,-0.94732)
3.02569	(-0.65457,-0.75600)	(-0.65327,-0.75712)
4.81555	(-0.85025,-0.52636)	(-0.87940,-0.47608)
7.46683	(-0.97271,-0.23202)	(-0.98976,-0.14269)
11.51978	(-0.99144,0.13054)	(-0.97761,0.21042)
18.08640	(-0.98124,0.19276)	(-0.83431,0.55128)
29.50770	(-0.55020,0.83502)	(-0.55638,0.83092)
51.46802	(-0.57022,0.82149)	(-0.17016,0.98541)
97.39943	(-0.60598,0.79547)	(0.22987,0.97322)
198.48350	(0.56937,0.82208)	(0.55214,0.83374)
443.84777	(-0.75306,0.65794)	(0.77729,0.62914)

图 6: 各个正特征能量 (E_{cm}) 下的相移

在能量 E_{cm} 约高于 50MeV 的时候, THO 给的相移和标准的 numerov 方法给的相移不能符合了。