



计算机科学与技术系

软件工程实验报告

实验名称: 等价判断功能实现

学号: 201180180

姓名: 李增基

实验日期: 2022.11.17

目录

1	git 基础操作	2
1.1	add	2
1.2	commit	2
1.3	reset	2
2	项目模块设计	6
2.1	项目环境	6
2.2	项目仓库	6
2.3	项目结构	6
3	等价判断功能实现	9
3.1	Program 类	9
3.2	Checker 类	9
3.3	TestGenerator 类	10
4	实验运行结果	10
5	git 拓展操作	11
5.1	rebase	11
5.2	revert	11
5.3	stash	12

1 git 基础操作

本节内容参考了<https://git-scm.com/docs/gittutorial>。

1.1 add

`git add` 命令主要用于将文件从工作区添加到暂存区，具体功能参见<https://git-scm.com/docs/git-add>。运行结果如图 1所示。



```
~/SE/git-learning
> git init
Initialized empty Git repository in /home/zkr/SE/git-learning/.git/

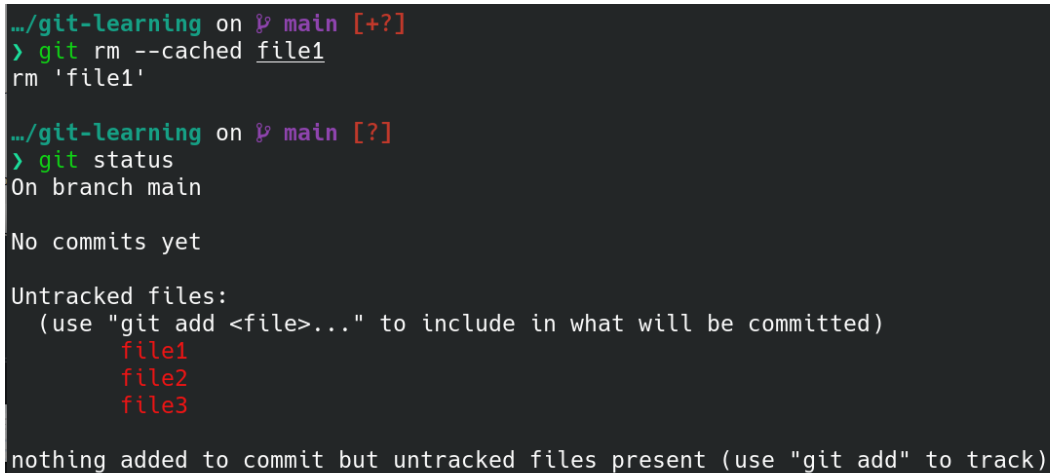
~/git-learning on ̷ main
> touch file1 file2 file3

~/git-learning on ̷ main [?]
> git add file1

~/git-learning on ̷ main [?]
> git --no-pager diff --cached
diff --git a/file1 b/file1
new file mode 100644
index 0000000..e69de29
```

图 1: add

使用 `diff --cached` 查看当前暂存区状态，可以看到文件已经被添加到暂存区。可以使用 `git rm --cached` 命令将刚加入的 `file1` 从暂存区删除，如图 2所示。



```
~/git-learning on ̷ main [?]
> git rm --cached file1
rm 'file1'

~/git-learning on ̷ main [?]
> git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    file1
    file2
    file3

nothing added to commit but untracked files present (use "git add" to track)
```

图 2: rm

如果要将当前文件夹添加到暂存区，可以使用 `git add .` 命令。这次我将三个文件都添加到暂存区，如图 3所示。

1.2 commit

`git commit` 命令用于将暂存区的文件提交到本地仓库，具体功能参见<https://git-scm.com/docs/git-commit>。`-m` 参数用于指定本次提交的信息，如图 4所示。

也可以使用 `--allow-empty` 参数允许空的提交，如图 5所示。

对于已跟踪的文件，可以使用 `-a` 参数自动添加 `add` 过程，直接将所有修改提交到本地仓库，不需要手动 `git add`，如图 6所示。

1.3 reset

`git reset` 命令主要用于撤销操作，将 `HEAD` 恢复至指定版本，功能参见<https://git-scm.com/docs/git-reset>。

```
.../git-learning on ʘ main [?]  
> git add .  
  
.../git-learning on ʘ main [ +]  
> git status  
On branch main  
  
No commits yet  
  
Changes to be committed:  
  (use "git rm --cached <file>..." to unstage)  
    new file:   file1  
    new file:   file2  
    new file:   file3
```

图 3: add directory

```
.../git-learning on ʘ main [ +]  
> git commit -m 'commit 1'  
[main (root-commit) 73c8276] commit 1  
 3 files changed, 0 insertions(+), 0 deletions(-)  
 create mode 100644 file1  
 create mode 100644 file2  
 create mode 100644 file3  
  
.../git-learning on ʘ main  
> git status  
On branch main  
nothing to commit, working tree clean
```

图 4: commit

```
.../git-learning on ʘ main  
> git commit --allow-empty -m 'empty commit'  
[main 159ba0f] empty commit  
  
.../git-learning on ʘ main  
> git --no-pager log  
commit 159ba0f8625e1a936fe27afd714a644d49aa5a45 (HEAD -> main)  
Author: zoukiri-r480-debian <zoukiri180@gmail.com>  
Date:   Wed Nov 16 19:31:46 2022 +0800  
  
    empty commit  
  
commit 73c82764db2e37604c7b717a284d9c24dc36d043  
Author: zoukiri-r480-debian <zoukiri180@gmail.com>  
Date:   Wed Nov 16 19:30:42 2022 +0800  
  
    commit 1
```

图 5: empty commit

```
.../git-learning on ʘ main
> echo 'change 1\nchange 2' > file1

.../git-learning on ʘ main [!]
> git st
On branch main
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:   file1

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

.../git-learning on ʘ main [!]
> git commit -am 'commit 3'
[main 4508579] commit 3
 1 file changed, 2 insertions(+)

.../git-learning on ʘ main
> git --no-pager show
commit 4508579229329237cd5634959201d831b88be6b7 (HEAD -> main)
Author: zoukiri-r480-debian <zoukiri180@gmail.com>
Date:   Wed Nov 16 19:59:24 2022 +0800

    commit 3

diff --git a/file1 b/file1
index e69de29..8aab0ca 100644
--- a/file1
+++ b/file1
@@ -0,0 +1,2 @@
+change 1
+change 2
```

图 6: commit -a

我使用了 `git reset --soft` 命令，将 HEAD 恢复至指定版本，这里恢复到上一次提交的版本。如图 7 所示。

```
.../git-learning on ʘ main
> git reset --soft HEAD^

.../git-learning on ʘ main [+]
> git st
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   file1
```

图 7: reset soft

注意到此时暂存区和工作区的内容没有变化，因为使用了 `--soft` 参数，只是将 HEAD 恢复到指定版本。

如果要同时将工作区和暂存区的文件也同时恢复，可以使用 `git reset --hard` 命令，如图 8所示。

```
.../git-learning on ⌵ main
> git reset --hard HEAD^
HEAD is now at 159ba0f empty commit

.../git-learning on ⌵ main
> git st
On branch main
nothing to commit, working tree clean
```

图 8: reset hard

有关 `git reset` 的参数对暂存区、工作区的影响，可以参考图 9。

working index HEAD target				working index HEAD		
A	B	C	D	--soft	A	B D
				--mixed	A	D D
				--hard	D	D D
				--merge	(disallowed)	
				--keep	(disallowed)	

working index HEAD target				working index HEAD		
A	B	C	C	--soft	A	B C
				--mixed	A	C C
				--hard	C	C C
				--merge	(disallowed)	
				--keep	A	C C

working index HEAD target				working index HEAD		
B	B	C	D	--soft	B	B D
				--mixed	B	D D
				--hard	D	D D
				--merge	D	D D
				--keep	(disallowed)	

working index HEAD target				working index HEAD		
B	B	C	C	--soft	B	B C
				--mixed	B	C C
				--hard	C	C C
				--merge	C	C C
				--keep	B	C C

图 9: reset parameters

可见 `--soft` 永远不会改变暂存区和工作区的内容，而`--hard` 总是将暂存区和工作区的内容恢复到指定版本。

2 项目模块设计

2.1 项目环境

本实验项目完全使用 Java 实现，本机环境为 Debain 11, OpenJDK 19, gcc 10.2.1
开发环境为 IntelliJ IDEA 2022.2.3, 构建工具为 IntelliJ 内置。

2.2 项目仓库

本实验仓库公开在<https://github.com/ZoukiLi/nju-se-lab>

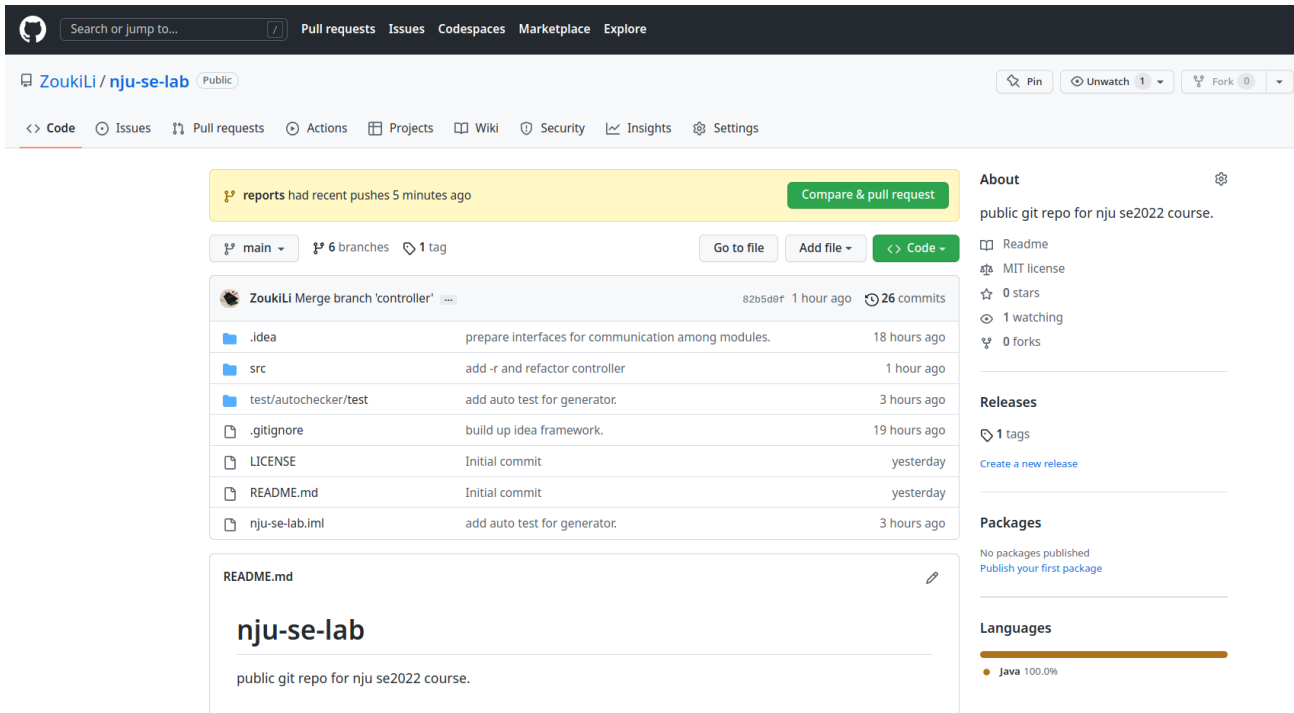


图 10: project repo

目前实验仓库有 6 个分支: `main`, `checker`, `reader`, `writer`, `controller`, `reports`

其中 `main` 是主分支, 不直接提交内容, 而是 `merge` 其他分支的改变, 我在 `main` 下 `merge` 时添加了 `--no-ff` 确保创建新的 `commit` 节点。

`checker`, `reader`, `writer`, `controller` 对应项目的 4 个子模块, 将在后文详细介绍。

`reports` 用来添加实验报告相关信息。

当前的提交记录图如图 11所示。

2.3 项目结构

项目的结构如图 12所示, 主要包含 4 个子模块: `autochecker`, `src_reader`, `writer`, `controller`。

`autochecker` 用于所有和生成程序有关的内容, 下包含一个子模块 `test`, 用于测试样例的生成。

`src_reader` 用于读取源文件, 需要实现抽象类 `SrcReader` 的功能。

`writer` 用于将比较结果写入文件, 需要实现抽象类 `ComparisonWriter` 的功能。

`controller` 用于控制整个程序的运行, 用来解析用户的参数, 同时生成了 `src_reader`, `writer`, 并将读入的结果用于调用 `autochecker` 的功能。

`main` 类只用于调用 `Controller` 和 `OptionParser`。

```

.../nju-se-lab on reports
> git --no-pager lg
* 1fc7a94 (HEAD -> reports, origin/reports) add manifest for package (3 minutes ago)
* 82b5d0f (tag: v0.0.1, origin/main, origin/HEAD, main) Merge branch 'controller' Add function: recursive
| \
| * 740457a (origin/writer, origin/reader, origin/controller, origin/checker, writer, reader, controller,
minutes ago) <zoukiri-r480-debian>
| /
* c7278df Merge branch 'writer' complete software part1: a.. (3 hours ago) <zoukiri-r480-debian>
| \
| * 67e4b80 add auto test for generator. (3 hours ago) <zoukiri-r480-debian>
| * cc938bc delete when exist output file. (3 hours ago) <zoukiri-r480-debian>
| * f22d28a bug fixed (3 hours ago) <zoukiri-r480-debian>
| * 6fa0771 Merge branch 'reader' into writer all parts were.. (11 hours ago) <zoukiri-r480-debian>
| \
| | * f11313d write reader (12 hours ago) <zoukiri-r480-debian>
| | * 6b78932 add writer (11 hours ago) <zoukiri-r480-debian>
| | * a8f6437 change Testclasses to Record (12 hours ago) <zoukiri-r480-debian>
| | * b1187d5 make Comparison Record, and add toString (12 hours ago) <zoukiri-r480-debian>
| | /
| | * 8561dd1 complete controller module (13 hours ago) <zoukiri-r480-debian>
| | * e3813a7 add some comments. (14 hours ago) <zoukiri-r480-debian>
| | * 7f3dd5c complete autochecker module (15 hours ago) <zoukiri-r480-debian>
| | * d50bd27 add TestRunningRecord to store running info (15 hours ago) <zoukiri-r480-debian>
| | * 29324b7 Merge branch 'reader' into checker SrcReader add.. (16 hours ago) <zoukiri-r480-debian>
| | \
| | | * 6175639 add copy method for SrcReader (16 hours ago) <zoukiri-r480-debian>
| | | * 5d13acf add @NotNull for SrcReader.fromPath (16 hours ago) <zoukiri-r480-debian>
| | | * a7e2d5e write auto checker (16 hours ago) <zoukiri-r480-debian>
| | | * 5d909da write Program Compare Interface (17 hours ago) <zoukiri-r480-debian>
| | | * a53097a write TestResult (17 hours ago) <zoukiri-r480-debian>
| | | * ec0e47b write TestGenerator (17 hours ago) <zoukiri-r480-debian>
| | | * 01b586d add submodule "test" for autochecker (18 hours ago) <zoukiri-r480-debian>
| | /
| | * 3d3136f prepare interfaces for communication among modul.. (18 hours ago) <zoukiri-r480-debian>
| | * 02402bf build up idea framework. (19 hours ago) <zoukiri-r480-debian>
| /
* 7c0d72f Initial commit (26 hours ago) <ZoukiLi>

```

图 11: git graph

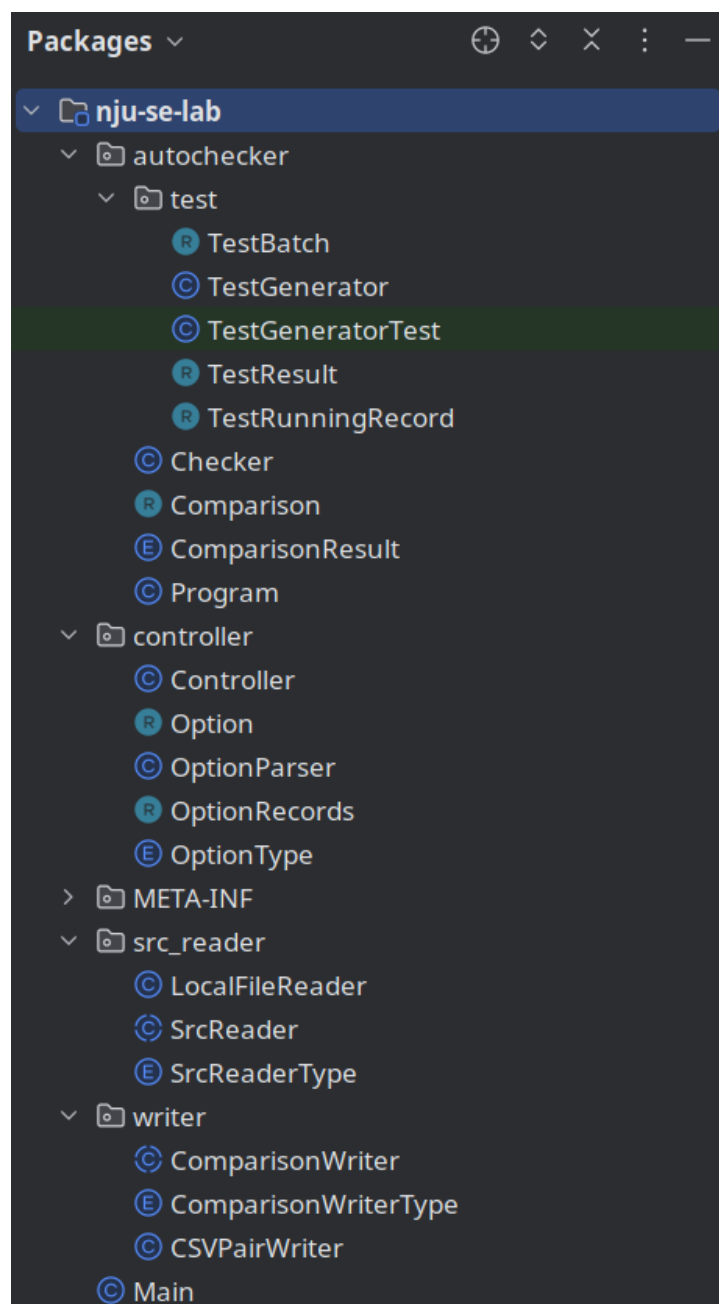


图 12: project structure

3 等价判断功能实现

3.1 Program 类

Program 类用于表示一个程序，包含了程序的源文件路径、编译结果。

其中最核心的问题是运行测试样例，我使用了 ProcessBuilder 来实现，ProcessBuilder 可以创建一个子进程，同时可以获得子进程的输入输出管道，方便向其中输入数据和输出数据。可以使用 waitFor 控制子进程的运行时间
核心代码如下，将运行结果保存到 records 中，注意 stdout 和 stderr 的我是分开保存：

```
1 // run the executable with the test batch
2 var records = new ArrayList<TestRunningRecord>();
3 testBatch.tests().forEach(t -> {
4     try {
5         var pb = new ProcessBuilder(_executablePath);
6         var runProcess = pb.start();
7         var stdin = runProcess.getOutputStream();
8         stdin.write(t.getBytes());
9         stdin.close();
10        boolean inTime = runProcess.waitFor(timeout, java.util.concurrent.TimeUnit.MILLISECONDS);
11        if (!inTime) {
12            runProcess.destroy();
13            records.add(new TestRunningRecord("", "", true));
14            return;
15        }
16        byte[] stdout = runProcess.getInputStream().readAllBytes();
17        byte[] stderr = runProcess.getErrorStream().readAllBytes();
18        records.add(new TestRunningRecord(new String(stdout), new String(stderr), false));
19    } catch (InterruptedException | IOException e) {
20        records.add(new TestRunningRecord("", "", true));
21    }
22 });
```

3.2 Checker 类

Checker 类用于根据输入路径产生一组 Program，并将其进行等价判断。

使用函数式编程的思想，Checker 很容易实现两两比较的功能，核心代码如下：

```
1 // compile all programs
2 _programs.forEach(Program::compile);
3 // generate a test batch
4 _testBatch = _testGenerator.generateTests();
5 // run the test batch on all programs
6 _programs.forEach(p -> p.runTests(_testBatch, timeout));
7 // compare all programs
8 _programs.forEach(p1 ->
9     _programs.stream().skip(_programs.indexOf(p1) + 1).forEach(p2 ->
10         _comparisons.add(new Comparison(p1, p2, p1.compare(p2))));
```

3.3 TestGenerator 类

TestGenerator 类用于生成测试样例。通过识别给定的 `stdin_format`, 产生一组测试样例, 可以使用正则简化代码, 核心代码如下:

```
1 var sb = new StringBuilder();
2 Arrays.stream(_fmt.split("\\r?\\n")).forEach(line -> {
3     Arrays.stream(line.split("\\s+")).forEach(type -> {
4         if (type.equals("char")) {
5             sb.append(randomAlpha());
6         } else {
7             var parts = type.split("[()],[\\s*");
8             if (parts[0].equals("int")) {
9                 sb.append(randomIntRange(Integer.parseInt(parts[1]), Integer.parseInt(parts[2])));
10            } else if (parts[0].equals("string")) {
11                sb.append(randomString(Integer.parseInt(parts[1]), Integer.parseInt(parts[2])));
12            }
13        }
14        sb.append(' ');
15    });
16    sb.deleteCharAt(sb.length() - 1);
17    sb.append('\\n');
18 });
```

4 实验运行结果

使用打包好的 jar 包运行, 参数及含义如下:

- `-i, --input` 指定输入文件夹, 默认为当前文件夹
- `-o, --output` 指定输出文件夹, 默认为当前文件夹
- `-r, --recursive` 是否递归查找输入文件夹下的子文件夹, 默认为 `false`
- `-t, --timeout` 指定超时时间, 默认为 `1000ms`
- `-n, --numTests` 指定测试用例数量, 默认为 `10`
- `-s, --seed` 指定随机种子, 默认为系统时间

选择 `java -jar nju-se-lab.jar -i input -r -o output` 运行, 结果如图 13 所示。输出的行数为 $94 = C_8^2 + C_{12}^2$ 所有输出文件在 `output` 文件夹下, 如下所示:

```
.../lab4-code on & main via v19
> java -jar nju-se-lab.jar -i input -r -o output

.../lab4-code on & main via v19 took 5s
> cat output/*.csv | wc -l
94

.../lab4-code on & main via v19
> head --lines=3 output/equal.csv
file1,file2
input/50A/138805414.cpp,input/50A/142890373.cpp
input/50A/138805414.cpp,input/50A/30534178.cpp
```

图 13: run

Listing 1: equal.csv

```
1 file1,file2
2 input/50A/138805414.cpp,input/50A/142890373.cpp
3 input/50A/138805414.cpp,input/50A/30534178.cpp
4 ...
5 input/4A/84822639.cpp,input/4A/173077807.cpp
```

Listing 2: inequal.csv

```
1 file1,file2
2 input/50A/138805414.cpp,input/50A/21508887.cpp
3 input/50A/138805414.cpp,input/50A/21508898.cpp
4 ...
5 input/4A/117364748.cpp,input/4A/48762087.cpp
```

5 git 拓展操作

这部分是我在实际中使用过的一些 git 操作，当时没有单独截图。

5.1 rebase

<https://git-scm.com/docs/git-rebase>

rebase 用来将 commit 信息重新生成到指定 base 上去，如果两者不在同一条提交线路上，会将 commit 信息重新生成一遍，附加到 base 上去。

与 merge 不同的是，merge 会将两条提交线路合并到一起，产生新的提交节点有多个源头，而 rebase 不会产生多个源头，而是将提交线路上的节点重新生成一遍，所以 rebase 产生的提交记录是线性的，不会像 merge 那样有网状结构。

我有的时候还会用 `git rebase -i` 来合并提交记录或修改之前几次的提交信息，但是需要注意的是，rebase 会产生新的提交节点，包括修改时间在内的元信息会发生变化。

5.2 revert

<https://git-scm.com/docs/git-revert>

`git revert` 用于生成一个新的提交记录，它所保存的修改内容刚好是指定提交记录的反向修改。，因此可以覆盖掉当时的提交记录。

与 `git reset` 不同的是，`git revert` 不会改变 `HEAD` 的指向，而是产生新的提交记录，刚好把之前的修改再改回来。

5.3 stash

<https://git-scm.com/docs/git-stash>

`git stash` 将当前工作区和暂存区的内容保存到一个栈中，可以通过 `git stash pop` 恢复。

`stash` 我在使用时主要是其他分支临时有个修改，而手头的工作不想提交，可以使用 `stash` 将本地修改保存起来，等到合并完成后再 `stash pop` 出来，恢复工作区的更改。