一， 思考题

1.1

(1) 如果小明的暂存区还有上次 add 过的 printf.c 或者 master 分支中 printf.c 没有被删除，那么使用

git checkout printf.c

即可恢复。

实验结果如下：

```
15061200@ubuntu:~/oslab/15061200-lab$ ls
boot     include     learnGit  modified.txt  testdata
drivers  include.mk  lib       printf.c      testdata~
gxemul   init        Makefile  Stage.txt     tools
15061200@ubuntu:~/oslab/15061200-lab$ rm printf.c
15061200@ubuntu:~/oslab/15061200-lab$ ls
boot     gxemul   include.mk  learnGit  Makefile      Stage.txt  testdata~
drivers  include  init        lib       modified.txt  testdata   tools
15061200@ubuntu:~/oslab/15061200-lab$ git checkout printf.c
15061200@ubuntu:~/oslab/15061200-lab$ ls
boot     include     learnGit  modified.txt  testdata
drivers  include.mk  lib       printf.c      testdata~
gxemul   init        Makefile  Stage.txt     tools
```

(2) 小红还没有将 rm 操作提交，因此就还可以通过撤销暂存区的 rm 操作并再次 checkout 使得文件回到本地，执行以下命令：

git reset HEAD printf.c

gti checkout printf.c

即可

```
15061200@ubuntu:~/oslab/15061200-lab$ git rm printf.c
rm 'printf.c'
15061200@ubuntu:~/oslab/15061200-lab$ ls
boot     gxemul   include.mk  learnGit  Makefile      Stage.txt  testdata~
drivers  include  init        lib       modified.txt  testdata   tools
15061200@ubuntu:~/oslab/15061200-lab$ git reset HEAD printf.c
Unstaged changes after reset:
D       README.txt
D       printf.c
15061200@ubuntu:~/oslab/15061200-lab$ git checkout printf.c
15061200@ubuntu:~/oslab/15061200-lab$ ls
boot     include     learnGit  modified.txt  testdata
drivers  include.mk  lib       printf.c      testdata~
gxemul   init        Makefile  Stage.txt     tools
```

或者直接使用

git checkout HEAD printf.c

```
15061200@ubuntu:~/oslab/15061200-lab$ ls
boot     include     learnGit  modified.txt  testdata
drivers  include.mk  lib       printf.c      testdata~
gxemul   init        Makefile  Stage.txt     tools
15061200@ubuntu:~/oslab/15061200-lab$ git rm printf.c
rm 'printf.c'
15061200@ubuntu:~/oslab/15061200-lab$ ls
boot     gxemul   include.mk  learnGit  Makefile      Stage.txt  testdata~
drivers  include  init        lib       modified.txt  testdata   tools
15061200@ubuntu:~/oslab/15061200-lab$ git checkout HEAD printf.c
15061200@ubuntu:~/oslab/15061200-lab$ ls
boot     include     learnGit  modified.txt  testdata
drivers  include.mk  lib       printf.c      testdata~
gxemul   init        Makefile  Stage.txt     tools
```

二者区别在于前者需要从暂存区检出，因此需要先撤销暂存区删除操作再检出，后者直接从尚未 commit 的目录树检出，检出后相关的删除请求也被删除。

(3) 只要使用

git rm –cached Tucao.txt

即可，然后再次 commit 就能将缓存区删除并且保留本地文件。

1.2

(1) 不全对，在 git clone 的时候默认会将所有分支克隆到本地（从 git clone 文档中得知），并且 checkout 执行的就是一个检出功能，但是检出哪一个分支是有我们可以决定的，加上参数即可决定需要检出哪一个分支如：

Git checkout lab1

除非我们不加任何参数如下

Git checkout

同时当前 head 指针指向 lab2

则我们会检出 lab2 分支

(2) 错，只要在指令后加上你所需要的分支，仍然会访问远程仓库，如下图：

```
15061200@ubuntu:~/oslab/15061200-lab$ git branch -r
  origin/lab1
  origin/lab1-result
  origin/lab2
15061200@ubuntu:~/oslab/15061200-lab$ git log remotes/origin/lab1 -n 1
commit 8ae9117010e8c0ec0a9c7cfdd674dcf92efcc5dc
Author: lmx <871628347@qq.com>
Date:   Tue Mar 21 14:26:04 2017 +0800

    15061200
```

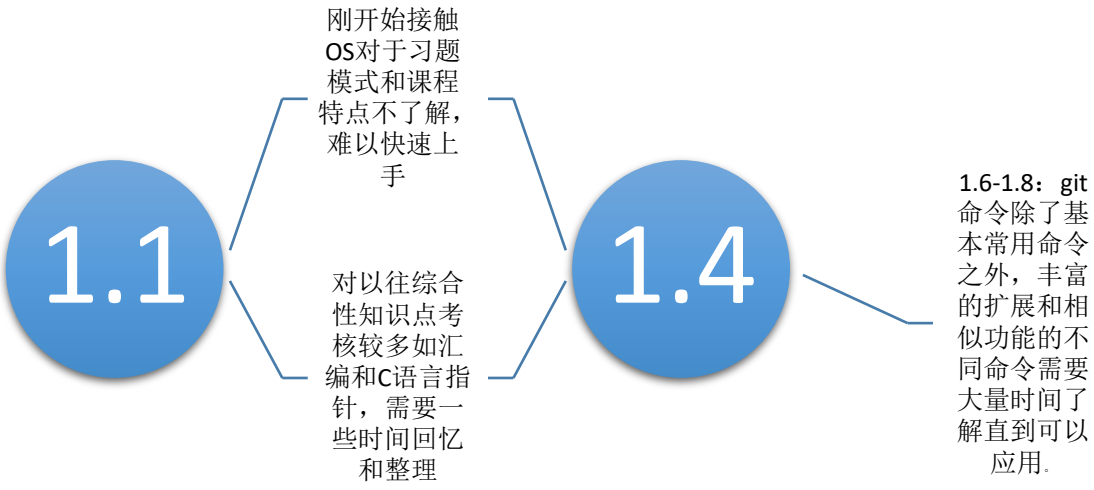
(3) 错，在(1)中已经阐明，git clone 默认会复制所有分支，除非指定特定的分支如下

git clone -b <branch> <remote_repo>

才可以只 clone 在指定仓库中的某个分支

(4) 正确，经过实验当 clone 之后输入 git branch 会显示当前正处在 master 分支上。

二， 实验难点图示



三， 体会与感想

本次 lab1 最大的难点在于刚接触操作系统实验，对于一个结合了 linux 操作，mips 内核，汇编语言和 C 语言的课程来说，我们前几个学期的综合学习成果和课外学习能力受到了严峻的挑战。

Lab1 里从 1.1 的寻找 mips-4KC 交叉编译器开始就如入五里雾中，但是我没有在群里看到有哪位同学问了或者讨论了这个问题，因此我一度觉得是自己不够认真看指导书导致的找不到，但翻来覆去的看还是没有结果，后来上 Unix 系统那节一般专业，才偶然想到可以用 locate 命令搜索前缀 mips-4KC 并成功找到了编译器位置。为此我后来也陆陆续续"采访"了不下二十多个同学，问他们是怎么找到的编译器，其中只有一个同学说使用搜索 locate 命令，另一个同学说用 find 权限不够，最后没办法问了室友答案，而剩下的十几个同学则全都是询问了朋友直接得到的答案。这不禁让我很疑惑，lab1 的 1.1 就让这么多同学迷茫,好好的练习最后变成了直接告诉答案，这中间反映出来的，我们的指导材料是不是还不够接地气，引导是不是还不够基础呢？

1.2 和 1.3 着重考察了一点点汇编语言的基础，上学期学好了计组的同学应该都没有什么大问题，但从我个人的体验和帮助舍友和几个朋友的经历中，我觉得大部分同学在习惯了计组 P7 之前的练习模式（给空填空，给要求做东西），突然面对这样开放，缺少参考资料（只有 guide book）而且课堂理论不怎么涉及 lab1 细节的情况下，去适应，去摸索，还是很有难度的，比如不少同学一开始其实是忘了写或者不知道还要写 j main 这句的。

1.4 的 print.c 着实让人难受了很久，其一是因为

当你刚看到 lp_Print() 的代码时，也许会手忙脚乱。

一下看不懂确实手忙脚乱；二是因为不敢相信前几个训练很友好的填空小题一下子就调到了这样的难度，而且 vim 的字体颜色和 config 文件对我而言学习成本有点划不来，用的很不顺手。问大神，大神说"很简单啊，你认真看啊"。刚开始做 print 的那个周五晚上着实心灰意冷。最后耐着性子分析代码结构，看懂了之后周六只用了两三小时

就完成了任务。

1.6-1.8 里面 Git 的学习主要是自己跟着指导书玩了玩，感觉 lab1 也就这一部分颇为亲切，没有露出为难你的狰狞脸孔。

总结来看，我的整个 lab1 实验应该花了十三个小时左右（1.1-1.6），个人感觉对于非大神的普通同学们来说，细节上难以理解或者快速上手，需要猜谜试错的成本太高，仍旧缺少指导。

四， 指导书反馈

结合第三部分的感想来看，主要有以下建议：

1. 猜谜试错成本太高，指导书对于刚上手的普通同学应该有更多细节上的照顾，比如大致提醒一下 linux 搜索或者课上再讲解一下 print
2. 考虑到 OO 的紧迫性，OS 应该在讲解上照顾一下当前实验，对于大部分人的疑难点进行讲解，或者直接添加实验相关的补充材料到 guide book 中
3. 有时候操作上的细节新手还不是很熟悉，每次经常有同学在群里问学长同样地问题，比如本地仓库 git clone & git checkout，比如修改文件之后再 make 一下然后再测试，这样的细节或许大神可以自己领悟，但是有的同学一旦卡住，就会因为这样的小细节浪费很多时间，因此建议指导书在***第一章***加入更多引路性质的内容。（***后期***可以根据难度保持一定的猜谜风格）
4. 虽然群里的大神们做得很快，但是也应该看到，系里无数同学通宵 OO 紧接着又通宵 OS，希望教学团队能够在课堂教学和进度上照顾一下落后的很多同学，比如组织做的好的同学帮助进行额外的补习或是加开实验答疑。

五， 残留难点

Lab1 主要残留疑点在于 git 命令的使用，比如撤销一个操作，网络答案五花八门，revert, add -i, reset, 回滚等各式各样的名词看得云里雾里。还需要时间练习体会。