CacheLab 实验说明 2023

实验内容

本实验分为 A/B 两部分, Part B 依赖于 Part A.

- Part A. 用 C 语言实现一个 cache simulator。
- Part B. 优化矩阵转置算法减少其 cache miss, 并利用 Part A 中实现的算法计算该算法的 cache miss 次数。

完成后请提交 `<SID>-handin.tar` 文件（该文件由 `make` 自动生成）和你的 **PDF** 格式实验报告两个文件. 实验报告中请描述 Part A 的实现架构设计与 Part B 的优化思路。

- 实验平台必须为Linux。
- ⚠️本次实验**强制要求**使用 `git` ，请**一定一定**要在实验过程中使用 `git`，这将包含一定的分数。
- 在最后的obe提交截止时间之前，**请保管好你自己的工作区所有文件**（除了你写的 `csim.c` 和 `trans.c`，其他文件都要保留，但**不需要提交到OBE**）。

实验步骤

获取实验文档

OBE平台上的压缩包中有实验说明文档，包括**本文档 / cachelab.pdf/ cachelab-ppt.pdf / Tutorial03_Git**.

- `cachelab.pdf` 是本次实验的第一手参考资料, 其中包含框架代码与测试接口的详细描述，以及编写代码时必须遵守的规则. 该文档包含你需要知道的关于本实验的所有信息. **务必仔细阅读** `cachelab.pdf`
- `cachelab-ppt.pdf` 为布置作业时使用的课件, 包含一些对 cache 的简介和实现实验简要思路。
- `Tutorial03_Git` 是 IPADS 新人培训 git 部分的PPT。
- `hacker` 在最后提交时需要用到，这个文件后续可能有改动，注意看通知。

获取实验文件

实验框架位于 `ics.ruc.rvalue.moe` 服务器的 `/mnt/ics1-2020/cachelab-handout.tar` 。 请自行将文件拷贝到自己的工作区并解包使用。 方法与先前实验一致在此不再赘述。

Part A

- Part A 部分需要你在 `csim.c` 文件中补全 `cache simulator` 的实现。
- 具体 I/O 格式以及测试数据的生成见 `cachelab.pdf` 。
- 参数选项解析可以自己实现，也可以使用 `getopt.h` ，详见 `cachelab-ppt.pdf`。
- 补全后在实验框架目录下执行 `make` 命令即可编译生成可执行程序 `test-csim` . 执行命令 `./test-csim` 即可测试 Part A 的正确性。

Part B

- Part B 部分需要你优化 `trans.c` 文件中的矩阵转置算法。
- 优化之后你可以执行 `make` 命令编译生成可执行程序 `test-trans`，执行命令 `./test-trans -M <m> -N  <n>` 即可测试 `trans.c` 中各个矩阵转置实现在 m 行 n 列矩阵的情况下的性能。
- 最终测试会使用 `./test-trans -M 32 -N 32` / `./test-trans -M 64 -N 64` / `./test-trans -M 61 -N 67` 三种情况进行测试。

最后提交

- `make` 会自动将你的 `csim.c` 和 `trans.c` 打包成 `<SID>-handin.tar`（`<SID>` 实际是你linux系统用户名，如果你在自己虚拟机上做实验，请改成u+学号），你需要在obe提交这个包。
- 你的实验报告，**pdf** 格式。
- 将 `hacker` 文件放入你**做实验的工作目录**下（就是你写代码所在的目录）：
    1. 用 `chmod 755 hacker`，将你的 `hacker` 文件修改为可执行。
    2. 执行 `./hacker <studen_id>`，其中`<student_id>` 是你的学号，例如执行 `./hacker 2023123456`，它将对 `csim.c` 和 `trans.c` 进行检查。
    3. 你可能会看到 `./hacker: /lib64/libcurl.so.4: no version information available (required by ./hacker)` ，可以忽略，确保输出如下信息：

        ```
        Hacking successfully.
        Reporting successfully.
        ```

        如果出现其他信息请联系助教。

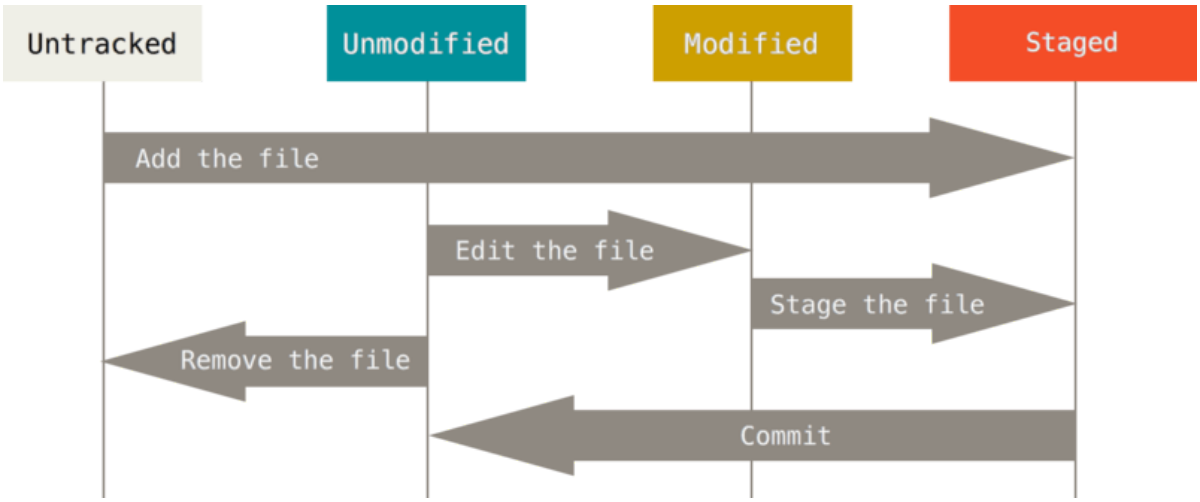
最终总评

`make` 之后执行 `./driver.py` 即可测试你在该实验的总体得分. 该测试脚本满分为 53 分, 不代表你在本实验的最终得分。

`make` 之后会在你的实验框架目录下生成一个 `<SID>-handin.tar` 文件. 请将其与你的实验报告一同提交至OBE。

Git简单使用

Git工作流



工作目录下的每一个文件都不外乎这两种状态：**已跟踪** 或 **未跟踪。**

已跟踪的文件是指那些被纳入了版本控制的文件，在上一次快照中有它们的记录，在工作一段时间后，它们的状态可能是未修改，**已修改**或**已放入暂存区。**简而言之，已跟踪的文件就是 Git 已经知道的文件。对于将暂存区提交就得到一个版本。

`git config` 设置

使用 `git config --global` 配置变量，包括用户名，邮箱，编辑器。

- `git config --global user.name 2019202120` ：请将用户名替换为你的学号。
- `git config --global user.email longtaifu@ruc.edu.cn` ：邮箱替换为你的邮箱。
- `git config --global core.editor vim` ：将编辑器改为 vim。

设置完成后用 `git config --global --list` 可以查看配置：

```
user.name=2019202120
user.email=longtaifu@ruc.edu.cn
core.editor=vim
```

`git init` 初始化你的仓库

首先进入你的工作目录，即tar包解压后的目录里面，执行 `git init` 或 `git init .` 来在该目录下新建一个仓库：

```
~/neospace/ics/cachelab-handout
❯ git init
Initialized empty Git repository in /mnt/ics1-2020/instructor/neospace/ics/cachelab-handout/.git/

~/neospace/ics/cachelab-handout master ?12
❯ ll
total 108
drwxr-xr-x.  4 instructor instructor  4096 Dec  8 10:49 ./
drwxrwxr-x. 12 instructor instructor  4096 Dec  8 10:47 ../
drwxr-xr-x.  7 instructor instructor  4096 Dec  8 10:49 .git/
-rw-r--r--.  1 instructor instructor   771 May  7  2014 Makefile
-rw-r--r--.  1 instructor instructor  1184 May  7  2014 README
-rw-r--r--.  1 instructor instructor  2138 Jan 22  2013 cachelab.c
-rw-r--r--.  1 instructor instructor  1033 Jul 10  2012 cachelab.h
-rwxr-xr-x.  1 instructor instructor 24904 Dec  7  2021 csim-ref*
-rw-r--r--.  1 instructor instructor    79 Jan 22  2013 csim.c
-rwxr-xr-x.  1 instructor instructor  4929 Nov  2  2012 driver.py*
-rwxr-xr-x.  1 instructor instructor 16848 Dec  7  2021 test-csim*
-rw-r--r--.  1 instructor instructor  8095 Jan 22  2013 test-trans.c
-rw-r--r--.  1 instructor instructor  2693 Jan 22  2013 tracegen.c
drwxr-xr-x.  2 instructor instructor  4096 Jun 13  2012 traces/
-rw-r--r--.  1 instructor instructor  2310 Jan 22  2013 trans.c
```

可以看到命令新建了一个 `.git` 目录，该目录就是仓库数据。

`git status` 查看当前状态

```
~/neospace/ics/cachelab-handout master ?12
❯ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       Makefile
#       README
#       cachelab.c
#       cachelab.h
#       csim-ref
#       csim.c
#       driver.py
#       test-csim
#       test-trans.c
#       tracegen.c
#       traces/
#       trans.c
nothing added to commit but untracked files present (use "git add" to track)
```

比如当前显示工作目录下有12个未被跟踪(untracked)的文件，并提示用 `git add` 来跟踪文件。

`git add <pathspec>` 将文件添加到暂存区

```
~/neospace/ics/cachelab-handout master ?12
❯ git add csim.c trans.c

~/neospace/ics/cachelab-handout master +2 ?10
❯ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   csim.c
#       new file:   trans.c
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       Makefile
#       README
#       cachelab.c
#       cachelab.h
#       csim-ref
#       driver.py
#       test-csim
#       test-trans.c
#       tracegen.c
#       traces/
```

这里我们把想要跟踪的文件添加到暂存区。例如本次实验需要写 `cism.c` 和 `trans.c` 两个文件，那么把两个文件添加到暂存区。执行后查看状态发现这两个文件变成了“未提交”。`<pathspec>` 也可以是目录，例如 `git add .` 把当前目录（即所有文件）添加到暂存区。

`git commit` 提交

使用 `git commit` 将暂存区的文件提交，命令会启动编辑器，输入本次提交的说明，保存退出后，完成提交：

```
This is a init commit.
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   csim.c
#       new file:   trans.c
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       Makefile
#       README
#       cachelab.c
#       cachelab.h
#       csim-ref
#       driver.py
#       test-csim
#       test-trans.c
#       tracegen.c
#       traces/
~
```

`#` 后面的内容是注释，不会作为提交信息保存。

```
~/neospace/ics/cachelab-handout master +2 ?10
❯ git commit
[master (root-commit) 6e92f7d] This is a init commit.
 2 files changed, 91 insertions(+)
 create mode 100644 csim.c
 create mode 100644 trans.c
```

你也可以直接使用 `git commit -m <msg>` 跳过编辑器，例如 `git commit -m "This is a init commit"`

修改文件

```
#include "cachelab.h"

int main()
{
    printSummary(0, 0, 0);
    return 0;
}
~
```

```
#include "cachelab.h"
#include <stdio.h>
int main()
{
    printf("Hello world!\n");
    printSummary(0, 0, 0);
    return 0;
}
~
```

这里对 `csim.c` 进行了修改，用 `git diff` （git diff其他用法参见git官方文档） 可以查看未暂存文件的更新：

```
~/neospace/ics/cachelab-handout master !1 ?10
❯ git diff
diff --git a/csim.c b/csim.c
index 44fdd4b..ac7d5da 100644
--- a/csim.c
+++ b/csim.c
@@ -1,7 +1,8 @@
 #include "cachelab.h"
-
+#include <stdio.h>
 int main()
 {
+    printf("Hello world!\n");
     printSummary(0, 0, 0);
     return 0;
 }
```

将修改后的文件添加到暂存区并提交，可以用 `git log` 查看提交记录：

```
~/neospace/ics/cachelab-handout master !1 ?10
❯ git add csim.c

~/neospace/ics/cachelab-handout master +1 ?10
❯ git commit -m "add print hello to csim.c"
[master 7d87c96] add print hello to csim.c
 1 file changed, 2 insertions(+), 1 deletion(-)

~/neospace/ics/cachelab-handout master ?10
❯ git log
commit 7d87c9631102057ad03cb1d5a22f8b52c729b113
Author: 2019202120 <longtaifu@ruc.edu.cn>
Date:   Fri Dec 8 14:55:27 2023 +0800

    add print hello to csim.c

commit 6e92f7dd9bf62fa327703b15f5f9493f2281d55a
Author: 2019202120 <longtaifu@ruc.edu.cn>
Date:   Fri Dec 8 14:33:31 2023 +0800

    This is a init commit.
```

分支与合并

`git init` 后会默认创建主分支 `master` 。使用 `git branch` 创建新分支，并用 `git checkout` 进入该分支：

```
~/neospace/ics/cachelab-handout master ?10
❯ git branch newbranch

~/neospace/ics/cachelab-handout master ?10
❯ git checkout newbranch
Switched to branch 'newbranch'
```

在newbranch分支中修改 `csim.c` 并提交：

```
┌─ ~/neospace/ics/cachelab-handout newbranch +1 ?10 ──
└─❯ git commit -m "changes in csim.c"
[newbranch 3018b25] changes in csim.c
 1 file changed, 3 insertions(+), 2 deletions(-)

┌─ ~/neospace/ics/cachelab-handout newbranch ?10 ─────
└─❯ git log
commit 3018b2582d8aaa5ccbc71fe9c7ffcb4b237ab8fd
Author: 2019202120 <longtaifu@ruc.edu.cn>
Date:   Fri Dec 8 15:55:51 2023 +0800

    changes in csim.c

commit 6e92f7dd9bf62fa327703b15f5f9493f2281d55a
Author: 2019202120 <longtaifu@ruc.edu.cn>
Date:   Fri Dec 8 14:33:31 2023 +0800

    This is a init commit.

┌─ ~/neospace/ics/cachelab-handout newbranch ?10 ─────
└─❯ git checkout master
Switched to branch 'master'

┌─ ~/neospace/ics/cachelab-handout master ?10 ────────
└─❯ git log
commit 6e92f7dd9bf62fa327703b15f5f9493f2281d55a
Author: 2019202120 <longtaifu@ruc.edu.cn>
Date:   Fri Dec 8 14:33:31 2023 +0800

    This is a init commit.
```

现在可以看到 master 和 newbranch的不同。

此时 master 中的 `csim.c` :

```c
#include "cachelab.h"

int main()
{
    printSummary(0, 0, 0);
    return 0;
}
```

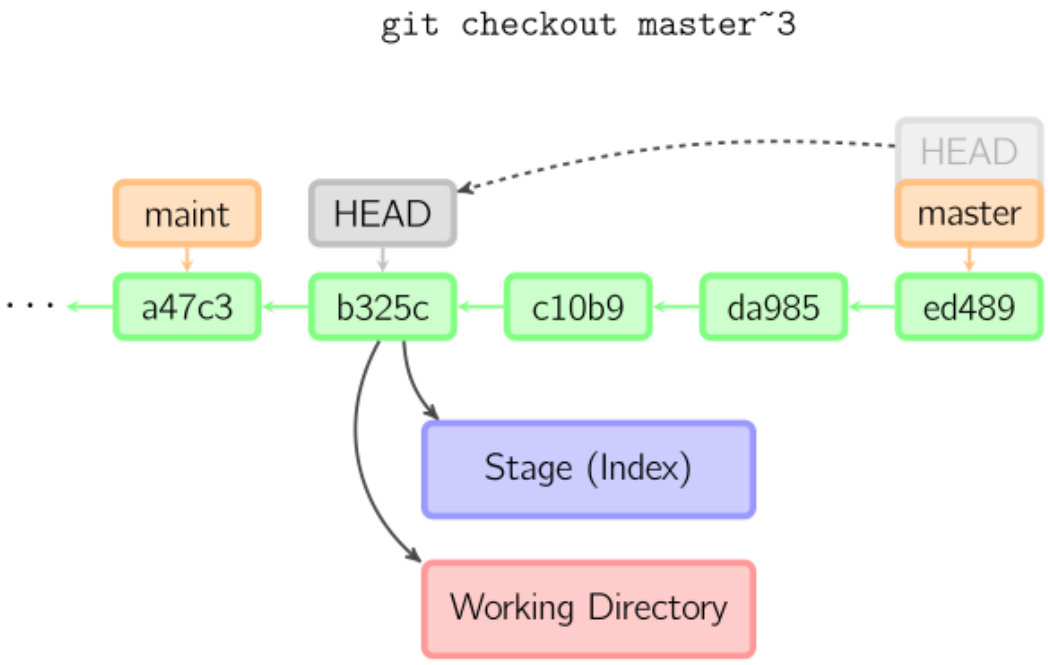
newbranch 中的 `csim.c`

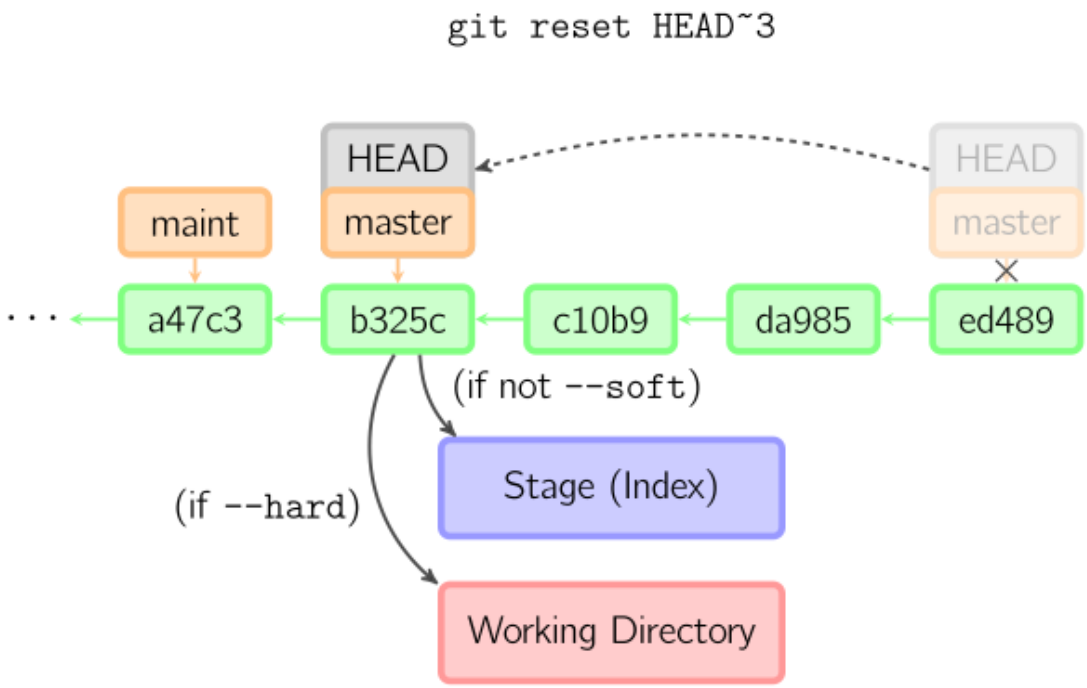
```c
#include "cachelab.h"
#include <stdio.h>
int main()
{
    printf("Hello world!");
    // This line deleted
    return 0;
}
```

使用 `git merge newbranch`，作用是把newbranch分支**合并到当前**(master)分支，可以看到master分支的 `csim.c` 变成了 newbranch 的版本：

```
~/neospace/ics/cachelab-handout master ?10
❯ git merge newbranch
Updating 6e92f7d..3018b25
Fast-forward
 csim.c | 5 +++--
 1 file changed, 3 insertions(+), 2 deletions(-)

~/neospace/ics/cachelab-handout master ?10
❯ cat csim.c
#include "cachelab.h"
#include <stdio.h>
int main()
{
    printf("Hello world!");
    // This line deleted
    return 0;
}

~/neospace/ics/cachelab-handout master ?10
❯ git log
commit 3018b2582d8aaa5ccbc71fe9c7ffcb4b237ab8fd
Author: 2019202120 <longtaifu@ruc.edu.cn>
Date:   Fri Dec 8 15:55:51 2023 +0800

    changes in csim.c

commit 6e92f7dd9bf62fa327703b15f5f9493f2281d55a
Author: 2019202120 <longtaifu@ruc.edu.cn>
Date:   Fri Dec 8 14:33:31 2023 +0800

    This is a init commit.
```

回溯版本





`git restore`

- `git restore <file>` ：将工作区文件回到最新一次提交的版本。
- `git restore --staged <file>`：将工作区文件回到当前暂存区的版本。
- `git restore --source=<commit> <file>`：将工作区文件回到指定的一次提交的版本。

例如最新一次提交文件内容为：

```
~/neospace/ics/cachelab-handout master ?11
❯ cat csim.c
#include "cachelab.h"
#include <stdio.h>
#include <stdlib.h>
int main()
{
    printf("Hello world!");
    // This line deleted
    return 0;
}
```

我们了修改文件，然后用 `git restore csim.c` 回到原来的版本：

```
~/neospace/ics/cachelab-handout master !1 ?11
❯ cat csim.c
#include "cachelab.h"
#include <stdio.h>
// deleted line
int main()
{
    printf("Hello world!");
    // This line deleted
    return 0;
}

~/neospace/ics/cachelab-handout master !1 ?11
❯ git restore csim.c

~/neospace/ics/cachelab-handout master ?11
❯ cat csim.c
#include "cachelab.h"
#include <stdio.h>
#include <stdlib.h>
int main()
{
    printf("Hello world!");
    // This line deleted
    return 0;
}

~/neospace/ics/cachelab-handout master ?11
❯
```

如果文件被删除也能同样恢复。

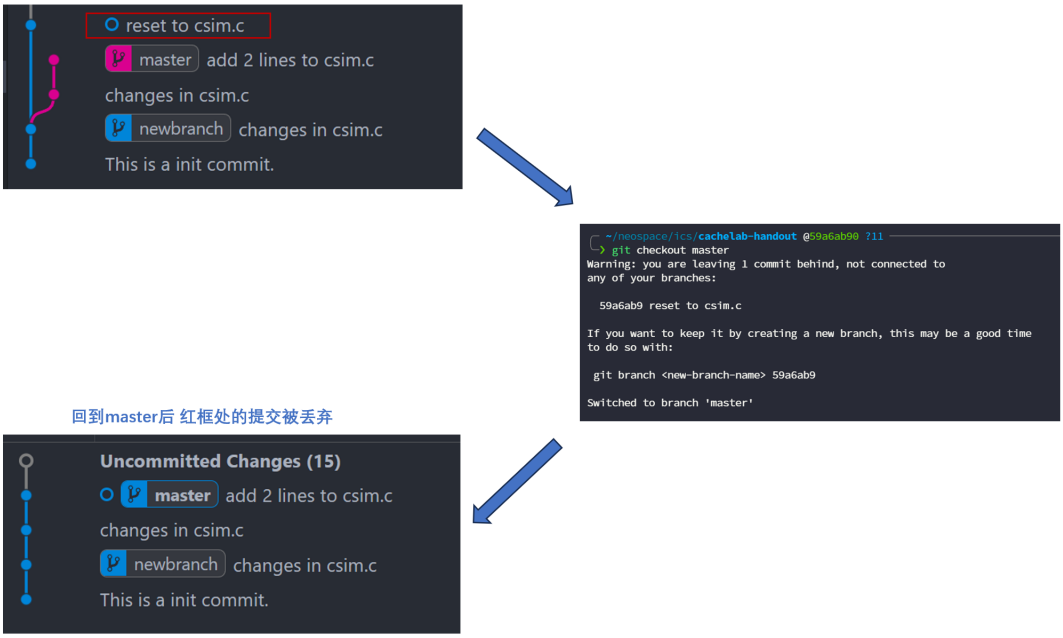
`git checkout <commit>`

使用 `git checkout <commit>` 可以**查看**以前的版本，`<commit>` 是某一次提交的hash值（hash值在git log中可以看到，可以不用写全，也可以用 `HEAD~i`， `HEAD` 代表当前头指针，一般指向最新一次提交i 表示 HEAD 的前 i 个提交，例如 `HEAD~2` 表示上上次提交），这样在工作区中你已跟踪的文件会回到该提交时的状态。

- ⚠️用 `git checkout` 回到某次提交这种方式仅限查看，**不要直接在这个状态进行提交**，你可以看到此时并不属于任何分支，你也可以看到执行 `git checkout` 执行后的提示：当前是 'detached HEAD' 的状态，你的所有修改和提交都不会影响其它分支（例如下图你直接提交实际不在master分支上可以理解为一个临时隐藏分支，而且你回到master你的这个临时分支会消失），因此你如果想

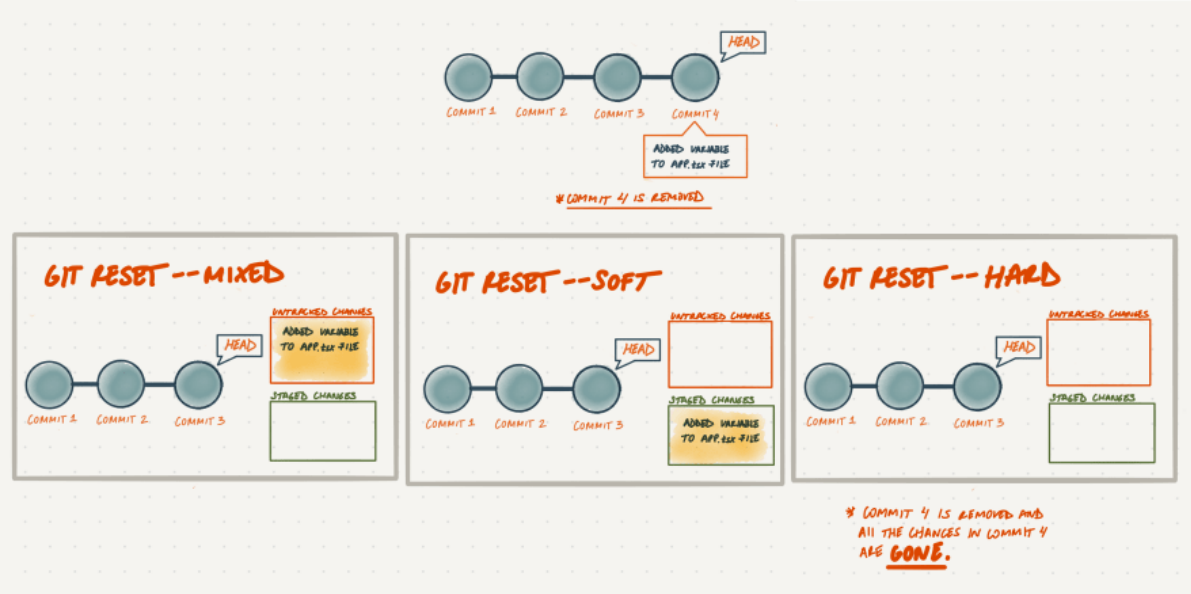
保留这个提交，你应该新建一个分支。

- 使用 `git checkout master` 回到原来 master 分支的状态（和 `HEAD` 一样 `master` 也是一个指针，指向master分支最后一次提交；不要用hash回到最新的提交，因为这样依然是 'detached HEAD'）。



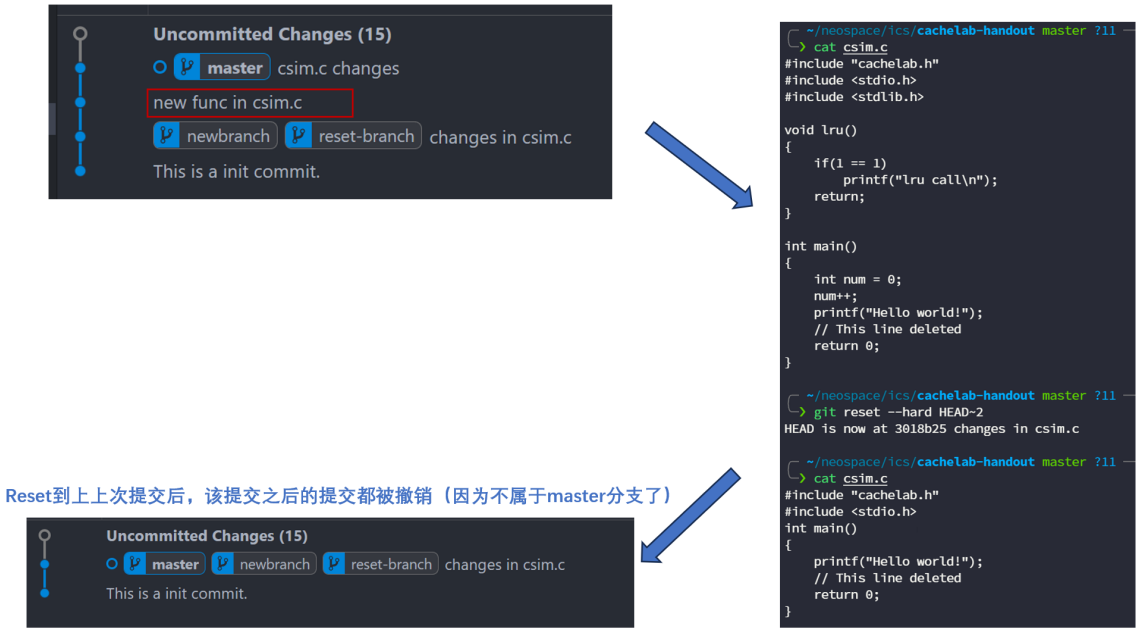
`git reset <commit>`

回滚提交，重置HEAD。



- `git reset --soft <commit>`：回到某次提交，不改变工作区和暂存区。
- `git reset --mixed <commit>`：回到某次提交，不改变工作区，但暂存区会变成目标提交时暂存区状态，也是`git reset` 不指定这三个选项的默认行为。
- `git reset --hard <commit>`：回到某次提交，工作区和暂存区都变成目标提交时的状态，（用之前考虑清楚）。

例如使用 `git reset --hard <commit>` 回到之前的某次提交并**删除该提交之后的所有提交数据**（实际可以恢复，但你最好这么理解）。



和 `git checkout` 不同，你可以在这之后进行后续的提交（你可以看到它依然是在 master 分支）。

推荐资料

- 官方中文文档 https://git-scm.com/book/zh/v2/
- 视频教程 https://www.bilibili.com/video/BV1r3411F7kn/?spm_id_from=333.337.search-card.all.click&vd_source=e14d1e1160b3a5d85de2977d9edb7211