

NEMU框架选讲(1): 编译运行

王慧妍

why@nju.edu.cn

南京大学



计算机科学与技术系



计算机软件研究所



提醒

PA1:

Deadline: 2022.10.9 23:59:59

Lab1:

Deadline: 2022.10.16 23:59:59

OJ今早更新上线，之前提交的同学麻烦重新提交

本讲概述

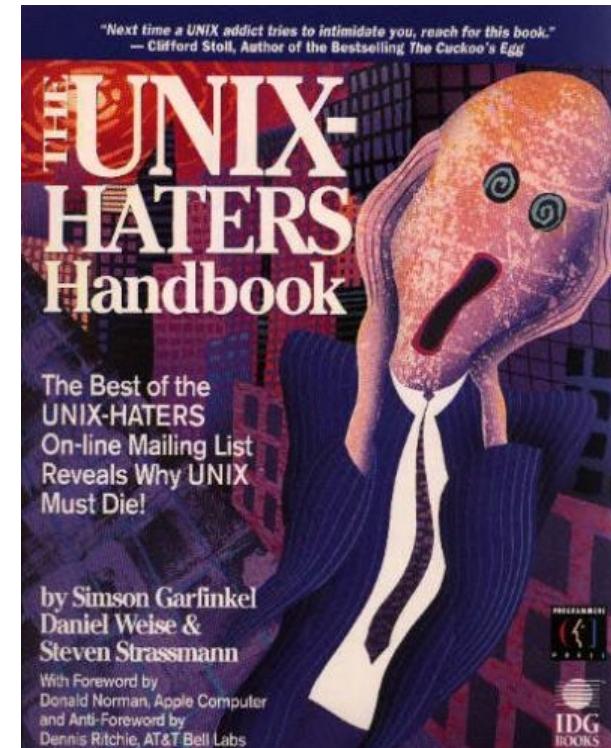
- Git, GitHub与代码仓库
 - Git选讲
 - Git在实验中的应用
- 项目的构建
 - Lab
 - NEMU
 - AbstractMachine

Git, GitHub和代码仓库

The UNIX Hater's Handbook (and Beyond)

- 写于1994年

- Simson Garfinkel 的主页有[电子版](#)
- 说有道理也有道理说没道理也没道理



UNIX , First Virus



Unix Myths

Unix Myths

Drug users lie to themselves. “Pot won’t make me stupid.” “I’m just going to try crack once.” “I can stop anytime that I want to.” If you are in the market for drugs, you’ll hear these lies.

Unix has its own collection of myths, as well as a network of dealers pushing them. Perhaps you’ve seen them before:

1. It’s standard.
2. It’s fast and efficient.
3. It’s the right OS for all purposes.
4. It’s small, simple, and elegant.
5. Shells and pipelines are great way to structure complex problems and systems.
6. It’s documented online.

7. It’s documented.
8. It’s written in a high-level language.
9. X and Motif make Unix as user-friendly and simple as the Macintosh.
10. Processes are cheap.
11. It invented:
 - the hierarchical file system
 - electronic mail
 - networking and the Internet protocols
 - remote file access
 - security/passwords/file protection
 - finger
 - uniform treatment of I/O devices.
12. It has a productive programming environment.
13. It’s a modern operating system.
14. It’s what people are asking for.
15. The source code:
 - is available
 - is understandable
 - you buy from your manufacturer actually matches what you are running.

You’ll find most of these myths discussed and debunked in the pages that follow.

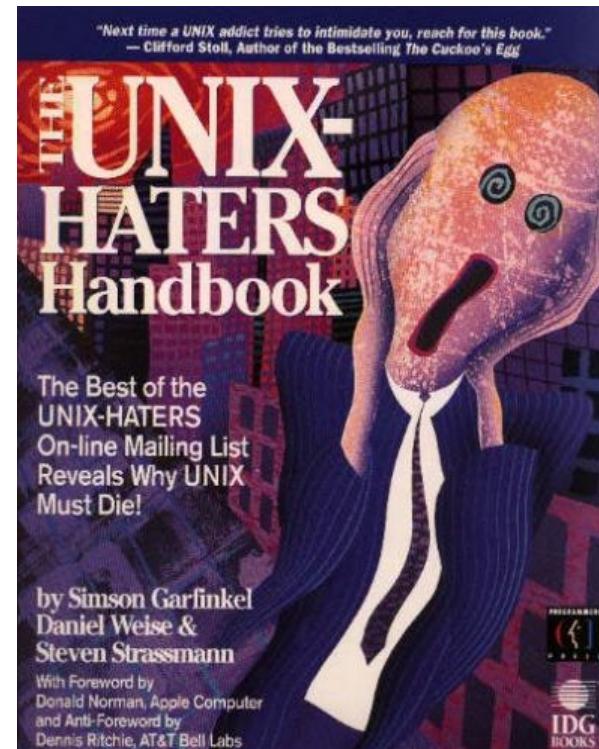


UNIX哲学

- *Keep it simple, stupid.* (KISS)
 - *Everything* is a file and *pipeline* programs to work together
- 一个命令只做 “一件事”
 - 从stdin输入 (printf)
 - 向stdout输出 (scanf)
 - 使用参数控制行为 (int main (int argc, char * argv[]);)
- 命令的输入和输出都是人类 + 机器均可读的文本
 - find .
 - wc -l a.txt b.txt
- 把命令的输入/输出连接起来 (管道) 协作完成任务
 - find . | grep '\.cpp\$' | xargs cat | wc -l

The UNIX Hater's Handbook (and Beyond)

- 写于1994年
 - Simson Garfinkel 的主页有电子版
 - 说有道理也有道理说没道理也没道理
- 至少指出了 UNIX 的一些缺陷
 - user friendly
 - 命令行/系统工具的缺陷
 - “rm” is forever
 - 手册的冗长
 - man wc
 - man gcc (26000+ 行)



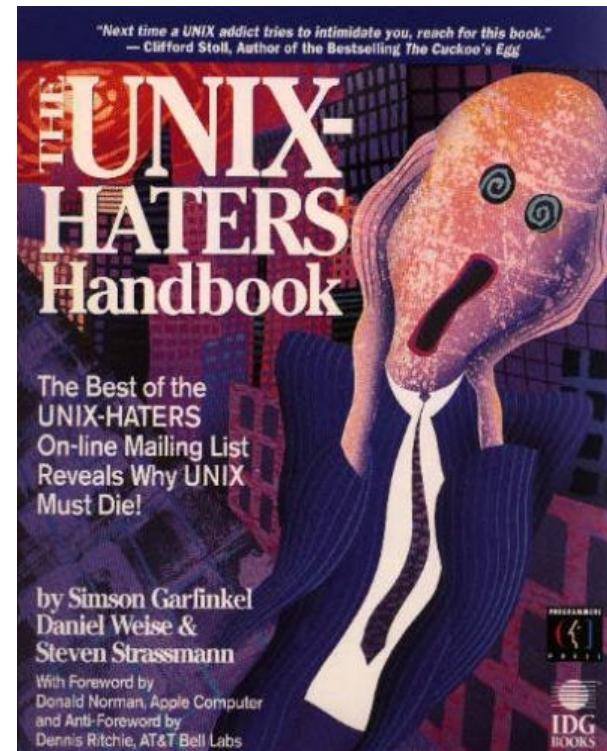
一个小例子

- rm -rf XX
 - 强制删除
 - 如果文件名叫-rf呢?
 - rm -rf?
 - rm -rf -rf?

```
$ ls -l
total 0
-rw-r--r-- 1 why why 0 Aug 25 18:10 -rf
$ rm -rf
$ ls -l
total 0
-rw-r--r-- 1 why why 0 Aug 25 18:10 -rf
$ rm -rf -rf
$ ls -l
total 0
-rw-r--r-- 1 why why 0 Aug 25 18:10 -rf
$ █
```

The UNIX Hater's Handbook (and Beyond)

- 写于1994年
 - Simson Garfinkel 的主页有电子版
 - 说有道理也有道理说没道理也没道理
- 至少指出了 UNIX 的一些缺陷
 - user friendly
 - 命令行/系统工具的缺陷
 - 手册的冗长
- 但今天 UNIX/Linux 已经成熟多了!
 - man
 - tldr



The Community Way

开源社区 (百度百科): 根据相应的开源软件许可证协议公布软件源代码的网络平台，同时也为网络成员提供一个自由学习交流的空间。

- 从GitHub获取代码
 - 传统工具链 + “现代” 编程体验
 - 有的时候网络不太靠谱



```
git clone -b 2022 git@github.com:NJU-ProjectN/ics-pa.git ics2022  
git clone https://github.com/NJU-ProjectN/ics-workbench
```



GitLab



<https://git.nju.edu.cn/>

The Community Way (cont'd)

GitHub is a development platform inspired by the way you work. From open source to business, you can host and review code, manage projects, and build software alongside 50 million developers.

- **无所不能的代码聚集地**

- 有整个计算机系统世界的代码
 - 硬件、操作系统、分布式系统、库函数、应用程序……

- **学习各种技术的最佳平台**

- 海量的文档、学习资料、博客（新世界的大门）
 - 提供友好的搜索（例子：`awesome C`）

学习Git?

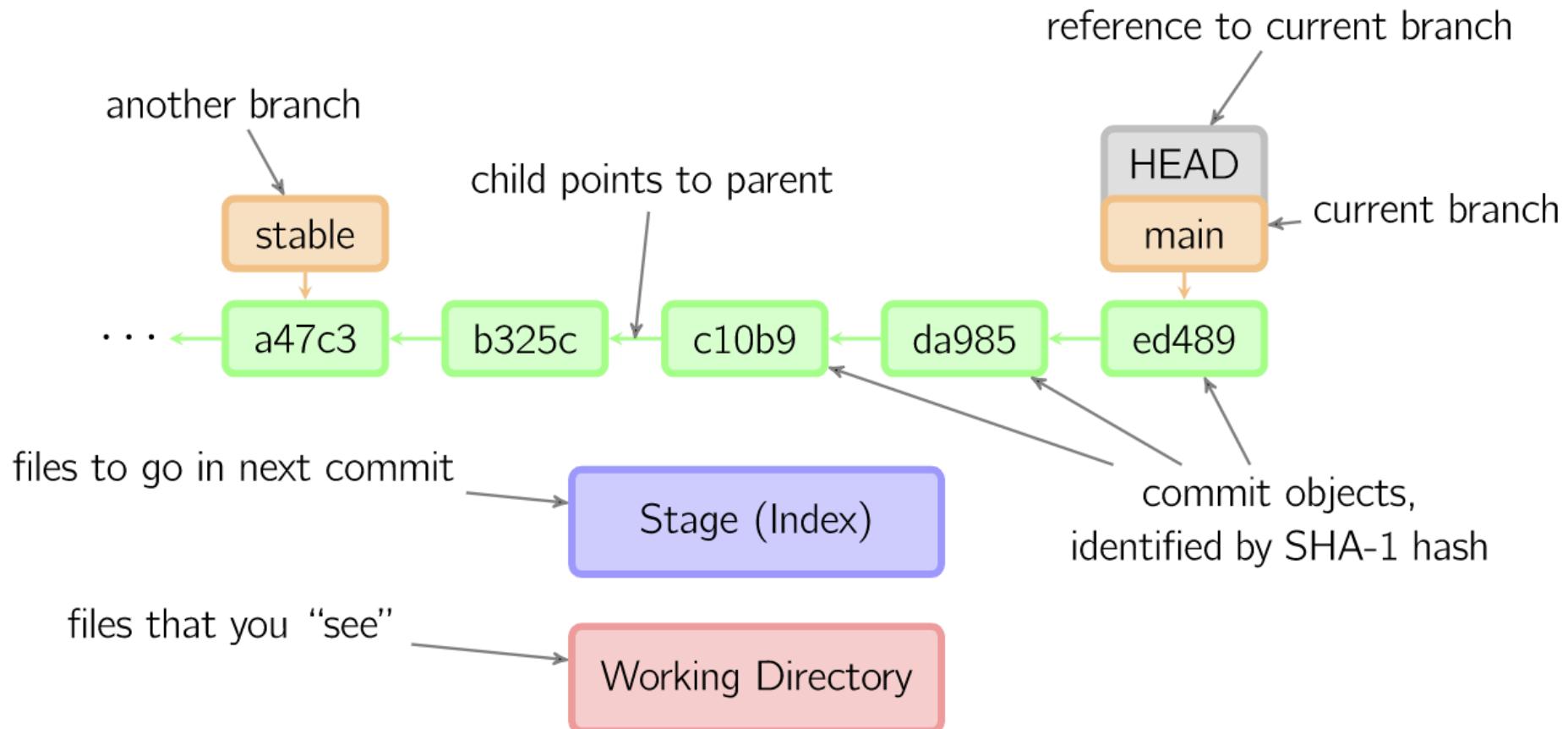
```
git clone -b 2021 https://github.com/NJU-ProjectN/ics-pa ics2021
```

内心独白：又要学新东西，我拒绝==

- RTFM?STFW!

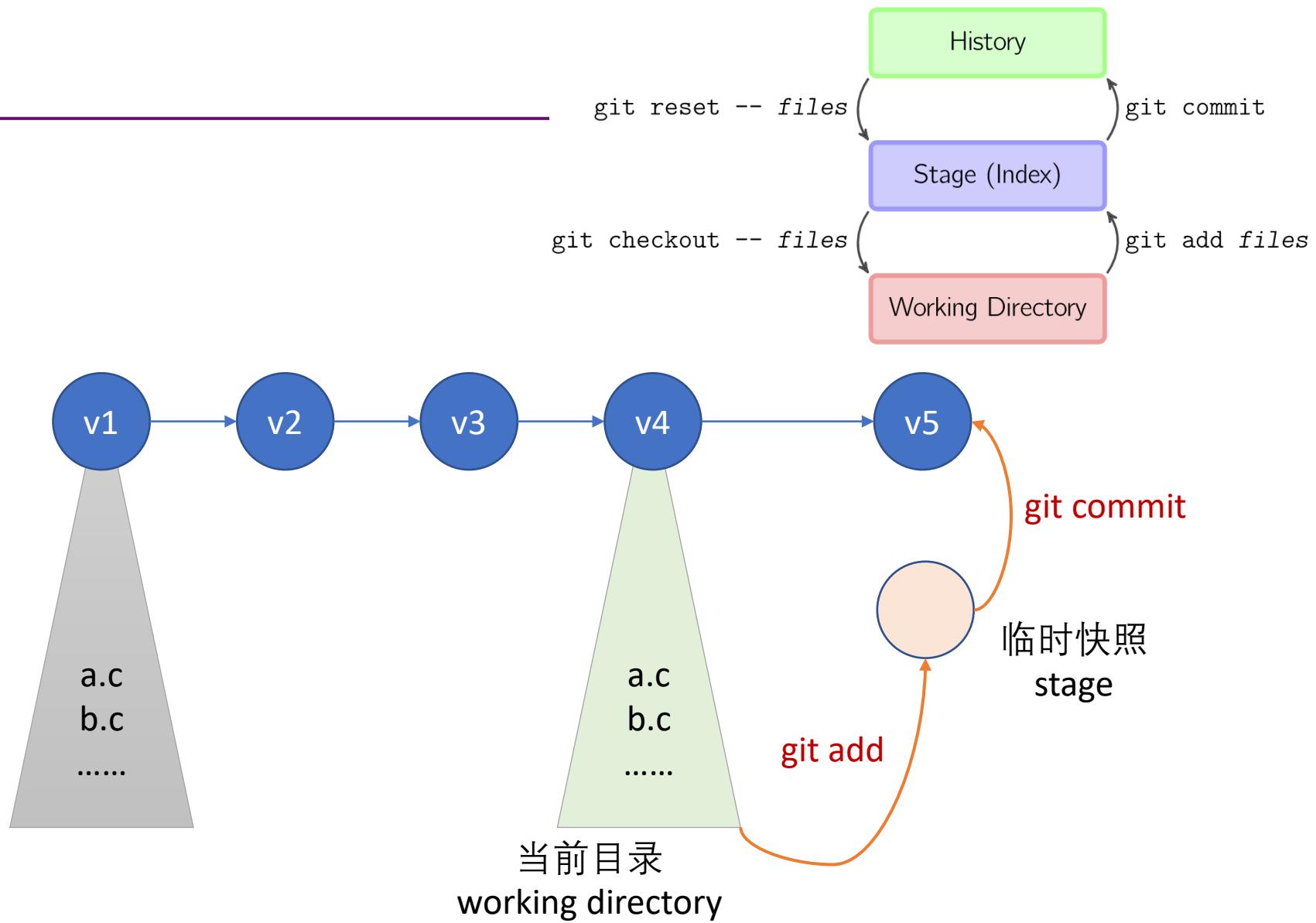
- 百度：得到一堆不太靠谱的教程
- 大家已经见识过**开源社区**的力量了
 - A Visual Git Reference
 - 英文、中文、日文.....
 - 好的文档是存在的
 - 还记得 tldr 吗？

A Visual Git Reference

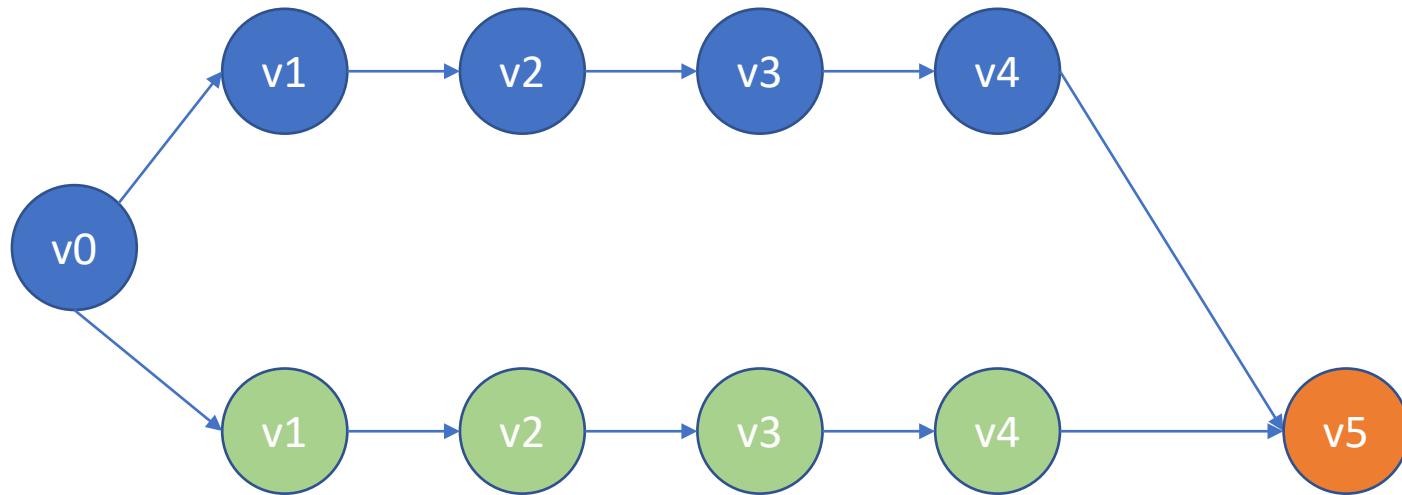


```
commit 8738b7b32e71a78f5b73376dc664780dd16800eb (HEAD -> pa1)
Author: tracer-ics2020 <tracer@njuics.org>
Date:   Thu Aug 19 18:27:15 2021 +0800
```

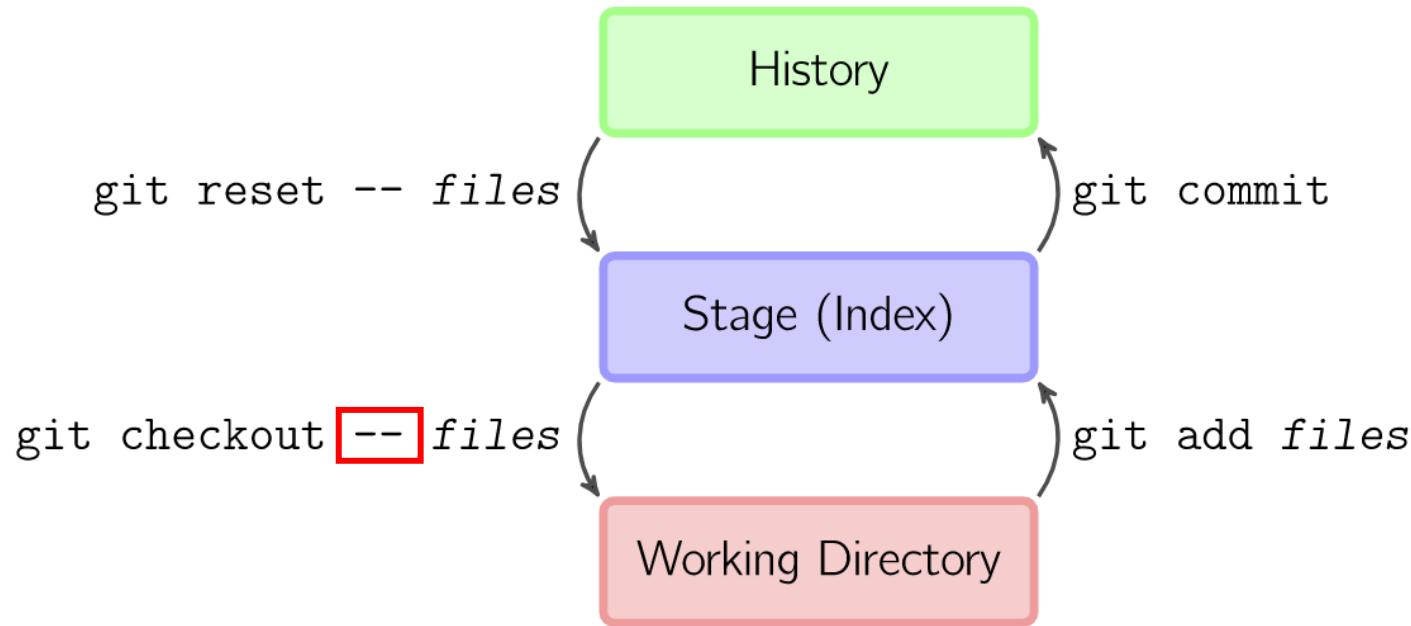
Git



Git：分布式版本控制系统



A Visual Git Reference (cont'd)



```
$ ls -l
total 0
-rw-r--r-- 1 why why 0 Aug 25 18:10 -rf
$ rm -- -rf
$ ls -l
total 0
```

一些Comments

- 有趣的 “--”
 - UNIX 的设计缺陷 (UGH 中点名批评)
 - 虽然是编程语言，但 Shell 更贴近自然语言
 - 也有很多 corner cases
 - 如果有一个文件叫 “-rf”……怎么删除它？？？
 - best practice: 文件名不以 “-” 开头、不含空格/符号……
- 体验 Git
 - 创建一个新的 repo，自由探索
 - 为什么 “预计完成时间 XX 小时” 是骗人的?
 - 预计完成时间是假设你在大一开始就用 Git 的
 - Visualizing Git Concepts with D3

Visualizing Git Concepts with D3

← → ⌂ ⚠ 不安全 | onlywei.github.io/explain-git-with-d3/#commit

Visualizing Git Concepts with D3

This website is designed to help you understand some basic git concepts visually. This is my first attempt at using both SVG and D3. I hope it is helpful to you.

Adding/staging your files for commit will not be covered by this site. In all sandbox playgrounds on this site, just pretend that you always have files staged and ready to commit at all times. If you need a refresher on how to add or stage files for commit, please read [Git Basics](#).

Sandboxes are split by specific git commands, listed below.

Basic Commands	Undo Commits	Combine Branches	Remote Server
git commit	git reset	git merge	git fetch
git branch	git revert	git rebase	git push
git checkout	git checkout -b		git pull
			git tag

Fork me on GitHub

We are going to skip instructing you on how to add your files for commit in this explanation. Let's assume you already know how to do that. If you don't, go read some other tutorials.

Pretend that you already have your files staged for commit and enter `git commit` as many times as you like in the terminal box.

Type git commit a few times.

\$ enter git command

Local Repository

Current Branch: master

e137e9b...

master

HEAD

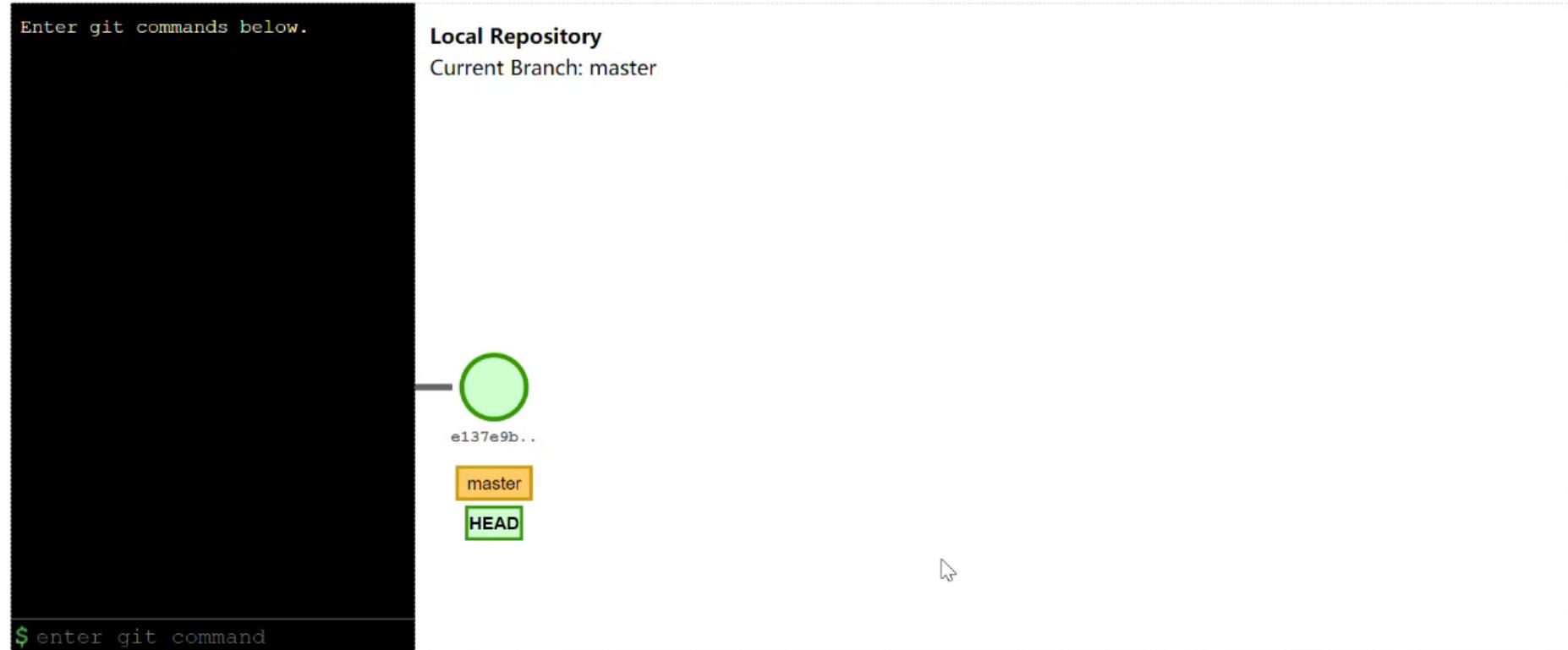
Specific Examples

Below I have created some specific real-world scenarios that I feel are quite common and useful.

[Restore Local Branch to State on Origin Server](#)
[Update Private Local Branch with Latest from Origin](#)
[Deleting Local Branches](#)

[Free Playground](#)
[Zen Mode](#)

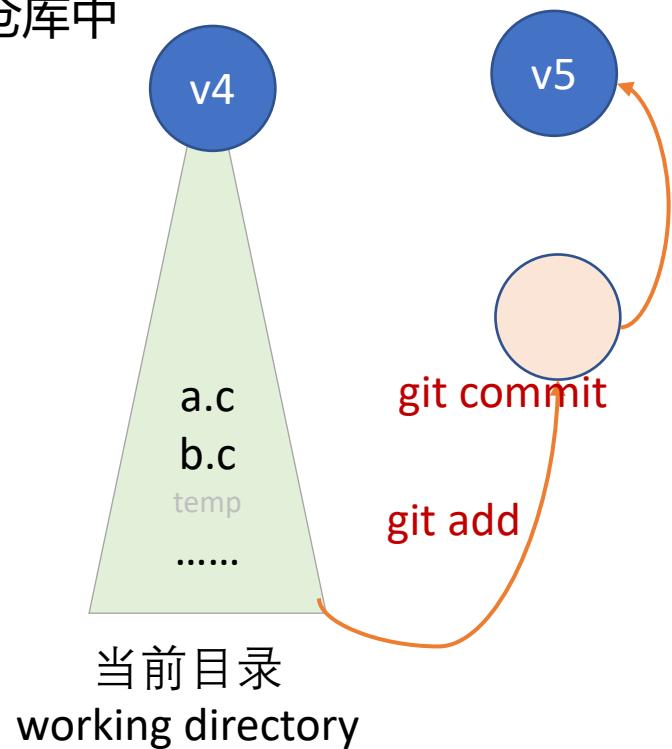
Visualizing Git Concepts with D3



一些Comments (cont'd)

- 我们使用了“白名单” .gitignore文件
 - 只在Git repo里管理.c, .h和Makefile
 - 基本原则：一切生成的文件都不放在Git仓库中

```
*          # 忽略一切文件  
!*/        # 除了目录  
!* .c      # .c  
!* .h      # ...  
!Makefile*  
.gitignore
```



- 为什么ls看不到这个文件?
 - 怎么还有一个.git

```
git clone -b 2021 https://github.com/NJU-ProjectN/ics-pa ics2021
```

获取PA代码

```
git clone -b 2022 git@github.com:NJU-ProjectN/ics-pa.git ics2022
```

- 获取来自github的一整个历史

提交脚本

- make submit会下载执行<http://jyywiki.cn/static/submit.sh>
 - Makefile

```
submit:
```

```
git gc  
STUID=$(STUID) STUNAME=$(STUNAME) bash -c “$(curl -s  
http://why.ink:8080/static/submit.sh)”
```



```
bash -c “$(curl -s http://why.ink:8080/static/submit.sh)”
```

- 我们做了什么?

```
why@why-VirtualBox:~/Documents/ICS2022/ics2022$ curl -s http://why.ink:8080/static/submit.sh  
COURSE=ICS2022  
MODULE=$(git rev-parse --abbrev-ref HEAD | tr '[a-z]' '[A-Z]')  
FILE=/tmp/upload.tar.bz2  
tar caf "$FILE" ".git" $(find . -maxdepth 2 -name "*.pdf") && \  
curl -F "token=$TOKEN" \  
    -F "course=$COURSE" \  
    -F "module=$MODULE" \  
    -F "file=@$FILE" \  
http://why.ink:8080/upload
```

提交脚本

- make submit会下载执行<http://why.ink:8080/static/submit.sh>

```
bash -c "$(curl -s http://why.ink:8080/static/submit.sh)"
```



```
# submit.sh (服务器)
```

```
COURSE=ICS2022
```

```
MODULE=$(git rev-parse --abbrev-ref HEAD | tr '[a-z]' '[A-Z]')
```

```
FILE=/tmp/upload.tar.bz2
```

PA1?PA2?

```
$ git branch
  master
* pa0
  pa1
$ git rev-parse --abbrev-ref HEAD
pa0
$ git checkout pa1
Switched to branch 'pa1'
$ git rev-parse --abbrev-ref HEAD
pa1
```

```
tar caf "$FILE" "$NAME/.git" $(find $NAME -maxdepth 2 -name "*.pdf") && \
curl -F "token=$TOKEN"
      -F "course=$COURSE"
      -F "module=$MODULE" \
      -F "file=@$FILE" http://why.ink:8080/upload
```

OJ得到的是.git

- .git包含了OJ想知道的一切

```
1 ./git  
2 ./git/objects  
3 ./git/objects/info  
4 ./git/objects/pack  
5 ./git/hooks  
6 ./git/info  
7 ./git/info/exclude  
8 ./git/HEAD  
9 ./git/branches  
10 ./git/description  
11 ./git/refs  
12 ./git/refs/heads  
13 ./git/refs/tags  
14  
15 ./git/config # 配置
```

Git追踪

- 另一段神秘代码 (来自 Makefile.lab)

```
git:  
@git add $(shell find . -name "*.c") \  
$(shell find . -name "*.h") -A --ignore-errors  
@while (test -e .git/index.lock); do sleep 0.1; done  
@(hostnamectl && uptime) | \  
git commit -F - -q --author=tracer-nju <tracer@nju.edu.cn>  
--no-verify --allow-empty @sync
```

- 在每次 make 执行时
 - git 目标都会被执行
 - 将 .c 和 .h 添加、强制提交到 Git repo

```
$ hostnamectl  
Static hostname: why-VirtualBox  
Icon name: computer-vm  
Chassis: vm  
Machine ID: 22e7c7921a0a437bb1817612c8ab2cce  
Boot ID: 51bcab304005479b8dcec150413ee8bf  
Virtualization: oracle  
Operating System: Ubuntu 21.04  
Kernel: Linux 5.11.0-34-generic  
Architecture: x86-64  
$  
$ uptime  
17:25:29 up 1:45, 1 user, load average: 0.11, 0.09, 0.02
```

```
6 GITFLAGS = -q --author='tracer-ics2021 <tracer@njuics.org>' --no-verify --allow-empty
7
8 # prototype: git commit(msg)
9 define git_commit
10   -@git add $(NEMU_HOME)/.. -A --ignore-errors
11   -@while (test -e .git/index.lock); do sleep 0.1; done
12   -@(echo "> $(1)" && echo $(STUID) && hostnamectl && uptime) | git commit -F - $(GITFLAGS)
13   -@sync
14 endef
```

```
commit 2662595b8712e2247fd1a92b5cb8b9103c9fc01b
Author: tracer-ics2021 <tracer@njuics.org>
Date: Thu Sep 16 12:07:21 2021 +0800

    > compile
ky2107911
        Static hostname: why-VirtualBox
        Icon name: computer-vm
        Chassis: vm
        Machine ID: 22e7c7921a0a437bb1817612c8ab2cce
        Boot ID: 6cbe073460244ae0ad90966cdbbe5961
        Virtualization: oracle
        Operating System: Ubuntu 21.04
            Kernel: Linux 5.11.0-34-generic
            Architecture: x86-64
12:07:21 up 2:40, 2 users, load average: 0.23, 0.23, 0.19

commit 713801b46b12063c27db811bd61a9b935395cd35
Author: tracer-ics2021 <tracer@njuics.org>
Date: Thu Sep 16 12:06:39 2021 +0800

    > run
ky2107911
        Static hostname: why-VirtualBox
        Icon name: computer-vm
        Chassis: vm
        Machine ID: 22e7c7921a0a437bb1817612c8ab2cce
        Boot ID: 6cbe073460244ae0ad90966cdbbe5961
        Virtualization: oracle
        Operating System: Ubuntu 21.04
            Kernel: Linux 5.11.0-34-generic
            Architecture: x86-64
12:06:39 up 2:39, 2 users, load average: 0.46, 0.26, 0.20
```

这是什么？

- 对抄袭代码的一种威慑
 - 如何抓抄袭
 - 旧方法；但至今仍在使用
 - Git 追踪是“一劳永逸的新方法”
 - 透过它，我们看到南大学子的人生百态



我们不生产~~水~~，我们只是~~大自然的搬运工~~
代码 互联网



农夫山泉®
NONGFU SPRING

代码抄袭：那些让985学生沉默，211学生流泪的真相



蒋炎岩 ✨
计算机科学与技术博士

2,221 人赞同了该文章

这是我们在TURC' 18 (SIGCSE China)的论文Needle: Detecting Code Plagiarism on Student Submissions [1]的科普版本。

“我们不生产代码，我们只是互联网的搬运工”——佚名

在平均学历是985的知乎，考不上个好学校都没脸出来冒泡。然而，到底是我们上了大学，还是大学上了我们？这篇文章来谈谈一个985学生知道了会沉默，211学生知道了会流泪的真相：广泛存在的抄作业问题。我想大部分读者朋友们看到这里的感受都是：中枪了。



心态崩了

你还记得上大学的时候，把“大腿”的作业拿来抄抄改改，然后去打游戏的日子吗？或者如果你就是同学们心目中的“大腿”，有没有慷慨相助同学们的经历？也许这类事情的确没有发生在你身上，但在计算机学科里，我们严肃的研究发现：

- 某985大学的一次实验大作业中，若不对代码抄袭控制，有超过82%的同学涉嫌拷贝代码(抄袭或被抄袭)，并且抄袭者会对代码做出不同程度的修改。大家也都知道，计算机专业课程光说不练课可就白上了，也就是说，在一届同学中有超过4/5这门课都白上了。考虑到还有同学选择不交作业，这个比例还会再高一些.....
- 在一所985大学的一门课程中，即便使用了抄袭检测工具、制定严格的惩罚措施、给有困难的同学提供“诚信分”，一门课程整个学期下来，依然有约20%的同学涉嫌拷贝代码。甚至有同学被逮到以后还会再抄、再被逮到以后再抄代码，铤而走险，屡教不改。

(以上结论仅针对我们调研的编程实验有效，而且导致同学们抄代码的原因是多种多样的，整体的代码抄袭情况我们尚未完全统计。)

这是什么？

- 对抄袭代码的一种威慑
 - 如何抓抄袭
 - 旧方法；但至今仍在使用
 - Git 追踪是“一劳永逸的新方法”
 - 透过它，我们看到南大学子的人生百态
- 记得 academic integrity
 - 我们留了足够多的分数给努力尝试过的同学通过
- 未来可能有终极加强版

小结：现代化的项目管理方式

- Git: 代码快照管理工具
 - 是一种 “可持久化数据结构”
 - 拓展阅读: Pro Git
- 框架代码中的两处非常规 Git 使用
 - 提交脚本
 - 仅上传 .git; 在服务器执行 git reset
 - 减少提交大小 (仅源文件)
 - Git 追踪
 - 编译时强制提交, 获取同学编码的过程
- 思考题: 如何管理自己的代码快照?
 - 提示: 分支/HEAD/... 只是指向快照的指针 (references)

项目构建

Make工具

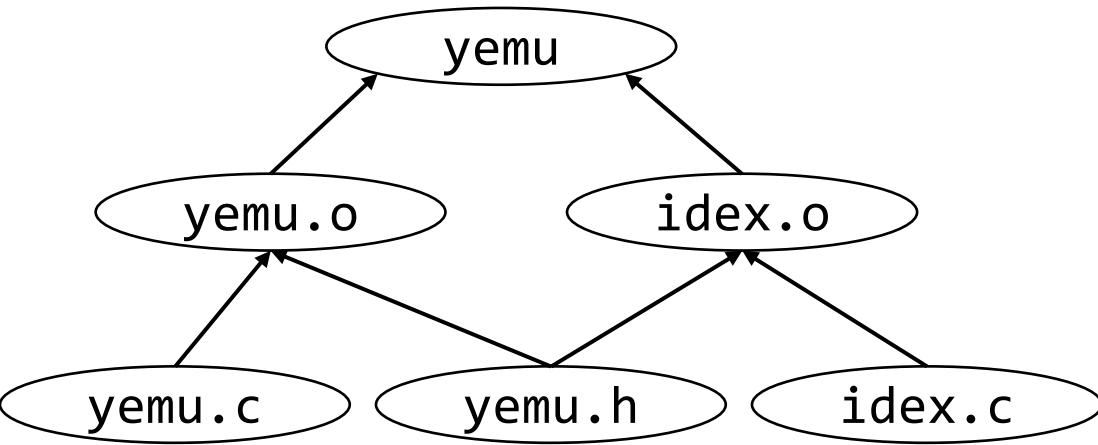
- 回顾：YEMU 模拟器
- Makefile 是一段 “declarative” 的代码
 - 描述了构建目标之间的依赖关系和更新方法

```
1 .PHONY: run clean test
2
3 CFLAGS = -Wall -Werror -std=c11 -O2
4 CC = gcc
5 LD = gcc
6
7 yemu: yemu.o idex.o
8     $(LD) $(LDFLAGS) -o yemu yemu.o idex.o
9
10 yemu.o: yemu.c yemu.h
11     $(CC) $(CFLAGS) -c -o yemu.o yemu.c
12
13 idex.o: idex.c yemu.h
14     $(CC) $(CFLAGS) -c -o idex.o idex.c
15
16 run: yemu
17     ./yemu
18
19 clean:
20     rm -f test yemu *.o
21
22 test: yemu
23     $(CC) $(CFLAGS) -o test idex.o test.c && ./test
```

Makefile

- make

- gcc -Wall -Werror -std=c11 -O2 -c -o yemu.o yemu.c
- gcc -Wall -Werror -std=c11 -O2 -c -o idex.o idex.c
- gcc -o yemu yemu.o idex.o

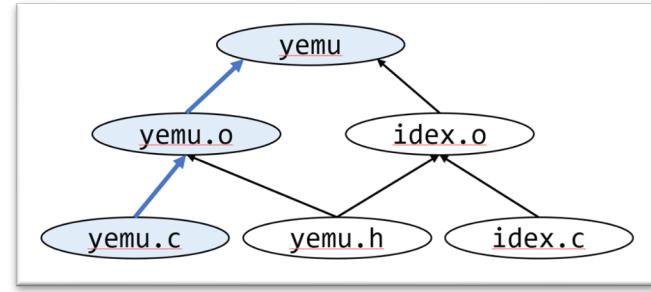
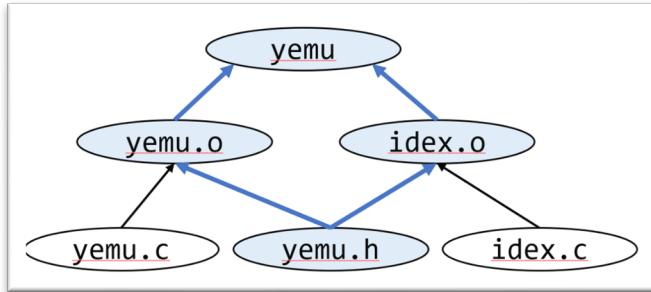


```
1 .PHONY: run clean test
2
3 CFLAGS = -Wall -Werror -std=c11 -O2
4 CC = gcc
5 LD = gcc
6
7 yemu: yemu.o idex.o
8     $(LD) $(LDFLAGS) -o yemu yemu.o idex.o
9
10 yemu.o: yemu.c yemu.h
11     $(CC) $(CFLAGS) -c -o yemu.o yemu.c
12
13 idex.o: idex.c yemu.h
14     $(CC) $(CFLAGS) -c -o idex.o idex.c
15
16 run: yemu
17     ./yemu
18
19 clean:
20     rm -f test yemu *.o
21
22 test: yemu
23     $(CC) $(CFLAGS) -o test idex.o test.c && ./test
```

Make工具

- 回顾：YEMU 模拟器
- Makefile 是一段 “declarative”的代码
 - 描述了构建目标之间的依赖关系和更新方法

```
$ make  
gcc -Wall -Werror -std=c11 -O2 -c -o yemu.o yemu.c  
gcc -Wall -Werror -std=c11 -O2 -c -o idex.o idex.c  
gcc -o yemu yemu.o idex.o  
$ make  
make: 'yemu' is up to date.
```



- make -j8

Make工具

- 回顾：YEMU 模拟器
- Makefile 是一段 “declarative” 的代码
 - 描述了构建目标之间的依赖关系和更新方法
 - 有用的编译选项
 - -nB、 -j
 - 同时也是和 Shell 结合紧密的编程语言
 - 能够生成各种字符串
 - 支持 “元编程” (#include, #define, ...)

Lab代码的构建

- 顶层 (top-level) Makefile:

```
# := -> C #define
NAME := $(shell basename $(PWD))
export MODULE := Lab1
```

```
# 变量 -> 字面替换
all: $(NAME)-64 $(NAME)-32
```

```
# include -> C #include
include ../Makefile
```

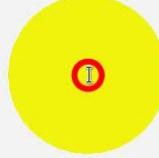
```
1 # **DO NOT MODIFY**
2
3 ifeq ($(NAME),)
4 $(error Should make in each lab's directory)
5 endif
6
7 SRCS := $(shell find . -maxdepth 1 -name "*.c")
8 DEPS := $(shell find . -maxdepth 1 -name "*.h") $(SRCS)
9 CFLAGS += -O1 -std=gnu11 -ggdb -Wall -Werror -Wno-unused-result -Wno-
unused-value -Wno-unused-variable
10
11 .PHONY: all git test clean commit-and-make
12
13 .DEFAULT_GOAL := commit-and-make
14 commit-and-make: git all
15
16 $(NAME)-64: $(DEPS) # 64bit binary
17     gcc -m64 $(CFLAGS) $(SRCS) -o $@ $(LDFLAGS)
18
19 $(NAME)-32: $(DEPS) # 32bit binary
20     gcc -m32 $(CFLAGS) $(SRCS) -o $@ $(LDFLAGS)
21
22 $(NAME)-64.so: $(DEPS) # 64bit shared library
23     gcc -fPIC -shared -m64 $(CFLAGS) $(SRCS) -o $@ $(LDFLAGS)
24
25 $(NAME)-32.so: $(DEPS) # 32bit shared library
26     gcc -fPIC -shared -m32 $(CFLAGS) $(SRCS) -o $@ $(LDFLAGS)
27
28 clean:
29     rm -f $(NAME)-64 $(NAME)-32 $(NAME)-64.so $(NAME)-32.so
30
31 include ../Makefile.lab
```

Applications Lab1: 大整数运算 - Mozilla... Terminal - why@debian... 18:11 拼写检查

File Edit View Terminal Tabs Help

Terminal - why@debian: ~/ICS_teach/ics-workbench/multimod

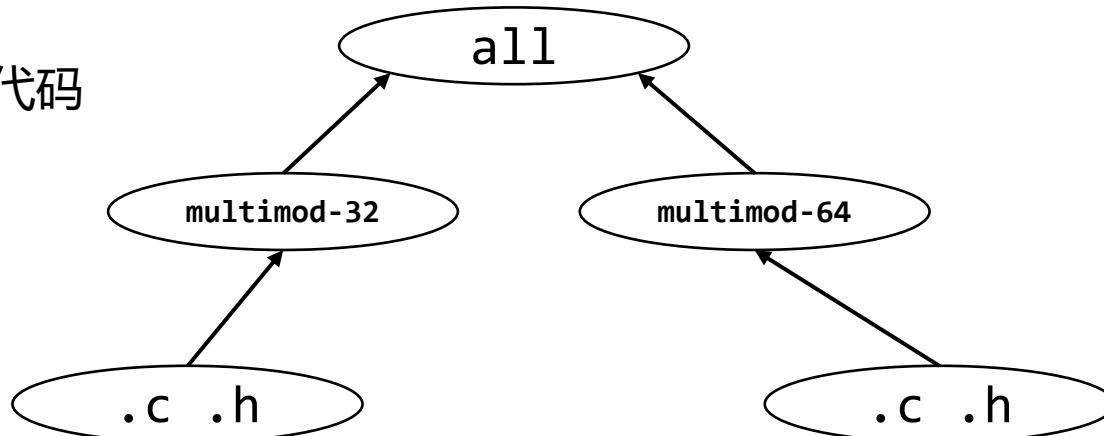
```
$ ls
main.c Makefile multimod-32 multimod-64 multimod.c
$ vim Makefile
$ vim Makefile
```



A large yellow circle is centered on the screen. Inside the circle is a smaller red circle containing the white letter 'I'.

Lab代码的构建 (cont'd)

- 构建目标
 - 总目标
 - .DEFAULT_GOAL := commit-and-make
 - commit-and-make: git all (all 在顶层 Makefile 中定义)
 - 可执行文件
 - multimod-64: gcc -m64
 - multimod-32: gcc -m32
 - 共享库 (之后的 lab 使用)
 - multimod-64.so: gcc -fPIC -shared -m64
 - multimod-32.so: gcc -fPIC -shared -m32
 - clean
 - 删除构建的代码



NEMU代码构建

- Makefile 真复杂
 - 放弃?

```
$ make
+ CC src/nemu-main.c
+ CC src/isa/riscv32/difftest/dut.c
+ CC src/isa/riscv32/system/intr.c
+ CC src/isa/riscv32/system/mmio.c
+ CC src/isa/riscv32/init.c
+ CC src/isa/riscv32/logo.c
+ CC src/isa/riscv32/reg.c
+ CC src/isa/riscv32/instr/decode.c
+ CC src/device/io/map.c
+ CC src/device/io/mmio.c
+ CC src/device/io/port-io.c
+ CC src/engine/interpreter/hostcall.c
+ CC src/engine/interpreter/init.c
+ CC src/cpu/difftest/dut.c
+ CC src/cpu/cpu-exec.c
+ CC src/monitor/sdb/watchpoint.c
+ CC src/monitor/sdb/sdb.c
+ CC src/monitor/sdb/expr.c
+ CC src/monitor/monitor.c
+ CC src/utils/log.c
+ CC src/utils/rand.c
+ CC src/utils/state.c
+ CC src/utils/timer.c
+ CC src/memory/paddr.c
+ CC src/memory/vaddr.c
+ LD /home/why/Documents/ICS2021/ics2021/nemu/build/riscv32-nemu-interpret
```

```
1 # Sanity check
2 ifeq ($(wildcard $(NEMU_HOME)/src/nemu-main.c),)
3   $(error NEMU_HOME=$(NEMU_HOME) is not a NEMU repo)
4 endif
5
6 # Include variables and rules generated by menuconfig
7 -include $(NEMU_HOME)/include/config/auto.conf
8 -include $(NEMU_HOME)/include/config/auto.conf.cmd
9
10 remove_quote = $(patsubst "%",%,$(1))
11
12 # Extract variables from menuconfig
13 GUEST_ISA ?= $(call remove_quote,$(CONFIG_ISA))
14 ENGINE ?= $(call remove_quote,$(CONFIG_ENGINE))
15 NAME    = $(GUEST_ISA)-nemu-$(ENGINE)
16
17 # Include all filelist.mk to merge file lists
18 FILELIST_MK = $(shell find ./src -name "filelist.mk")
19 include $(FILELIST_MK)
20
21 # Filter out directories and files in blacklist to obtain the final set of source files
22 DIRS_BLACKLIST-y += $(DIRS_BLACKLIST)
23 SRCS_BLACKLIST-y += $(SRCS_BLACKLIST) $(shell find $(DIRS_BLACKLIST-y) -name "*.c")
24 SRCS-y += $(shell find $(DIRS-y) -name "*.c")
25 SRCS = $(filter-out $(SRCS_BLACKLIST-y),$(SRCS-y))
26
27 # Extract compiler and options from menuconfig
28 CC = $(call remove_quote,$(CONFIG_CC))
29 CFLAGS_BUILD += $(call remove_quote,$(CONFIG_CC_OPT))
30 CFLAGS_BUILD += $(if $(CONFIG_CC_LTO),-flto,)
31 CFLAGS_BUILD += $(if $(CONFIG_CC_DEBUG),-ggdb3,)
32 CFLAGS_BUILD += $(if $(CONFIG_CC_ASAN),-fsanitize=address,)
33 CFLAGS += $(CFLAGS_BUILD) -D_GUEST_ISA_=$(GUEST_ISA)
34 LDFLAGS += $(CFLAGS_BUILD)
35
36 # Include rules for menuconfig
37 include $(NEMU_HOME)/scripts/config.mk
38
39 ifdef CONFIG_TARGET_AM
40 include $(AM_HOME)/Makefile
41 LINKAGE += $(ARCHIVES)
42 else
43 # Include rules to build NEMU
44 include $(NEMU_HOME)/scripts/native.mk
45 endif
```

~/Documents/ICS2021/ics2021/nemu/Makefile[1]

还是尝试去读一下

管理 控制 视图 热键 设备 帮助

```
why@why-VirtualBox:~/Documents/ICS2021/ics2021/nemu$ ls  
build configs include Kconfig Makefile README.md resource scripts src tools  
why@why-VirtualBox:~/Documents/ICS2021/ics2021/nemu$ vim Kconfig  
why@why-VirtualBox:~/Documents/ICS2021/ics2021/nemu$
```

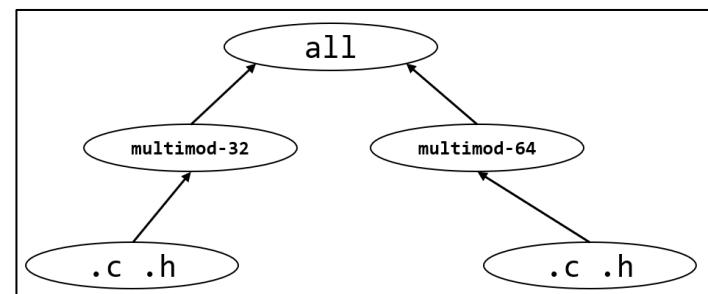
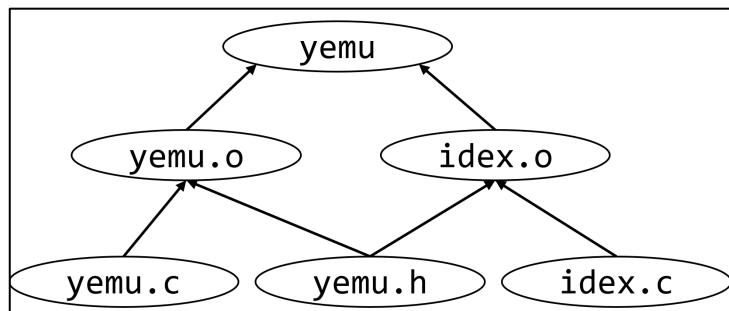
16

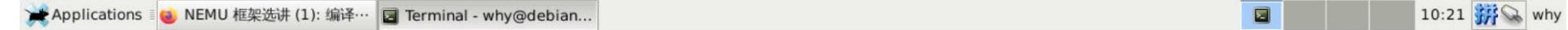
[conf] unix utf-8 Ln 1, Col 0/26

"why-Vi S 英 · 简拼 22-9月 -21

NEMU代码构建

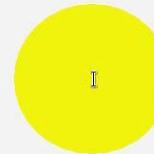
- Makefile 真复杂
 - 放弃?
- 一个小诀窍
 - 先观察 make 命令实际执行了什么 (trace)





Terminal - why@debian: ~/ics2021/nemu

```
$ ls
configs  include  Kconfig  Makefile  README.md  resource  scripts  src  tools
$ █
```



```

1 .DEFAULT_GOAL = app
2
3 # Add necessary options if the target is a shared library
4 ifdef SHARE
5 SO = -SO
6 CFLAGS += -fPIC
7 LDFLAGS += -rdynamic -shared -fPIC
8 endif
9
10 WORK_DIR = $(shell pwd)
11 BUILD_DIR = $(WORK_DIR)/build
12
13 INC_PATH := $(WORK_DIR)/include $(INC_PATH)
14 OBJ_DIR = $(BUILD_DIR)/obj-$(NAME)$(SO)
15 BINARY = $(BUILD_DIR)/$(NAME)$(SO)
16
17 CC ?= gcc
18
19 # Compilation flags
20 CC := $(CC)
21 LD := $(CC)
22 INCLUDES = $(addprefix -I, $(INC_PATH))
23 CFLAGS := -O2 -MMD -Wall -Werror $(INCLUDES) $(CFLAGS)
24 LDFLAGS := -O2 $(LDFLAGS)
25
26 OBJS = $(SRCS:.c=$(OBJ_DIR)/%.o)
27
28 # Compilation patterns
29 $(OBJ_DIR)/%.o: %.c
30   @echo + CC $<
31   @mkdir -p $(dir $@)
32   @$(CC) $(CFLAGS) -c -o $@ $<
33   $(call call_fixdep, $(@:o=d), $@)
34
35 # Dependencies
36 -include $(OBJS:.o=.d)
37
38 # Some convenient rules
39
40 .PHONY: app clean
41
42 app: $(BINARY)
43
44 $(BINARY): $(OBJS) $(ARCHIVES)
45   @echo + LD $@
46   @$(LD) -o $@ $(OBJS) $(LDFLAGS) $(ARCHIVES) $LIBS
47
48 clean:
49   -rm -rf $(BUILD_DIR)
~/Documents/ICS2021/ics2021/nemu/scripts/build.mk[2] [make] unix utf-8 Ln 46, Col 53/49
-- INSERT --
[0] 0:vim*

```

Applications NEMU 框架选讲 (1): 编译... Terminal - why@debian...

10:22 100% why

File Edit View Terminal Tabs Help

uthor='tracer-ics2021 <tracer@njuics.org>' --no-verify --allow-empty sync

echo + LD /home/why/ics2021/nemu/build/riscv32-nemu-interpreter

gcc -o /home/why/ics2021/nemu/build/riscv32-nemu-interpreter /home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/nemu-main.o /home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/engine/interpreter/init.o /home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/engine/interpreter/hostcall.o /home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/isa/riscv32/init.o /home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/isa/riscv32/system/intr.o /home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/isa/riscv32/system/mmu.o /home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/isa/riscv32/instr/decode.o /home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/isa/riscv32/logo.o /home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/isa/riscv32/reg.o /home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/cpu/cpu-exec.o /home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/cpu/difftest/dut.o /home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/monitor/monitor.o /home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/monitor/sdb/watchpoint.o /home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/monitor/sdb/sdb.o /home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/monitor/sdb/expr.o /home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/utils/log.o /home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/utils/state.o /home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/utils/timer.o /home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/utils/rand.o /home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/memory/paddr.o /home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/memory/vaddr.o /home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/device/io/map.o /home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/device/io/mmio.o /home/why/ics2021/nemu/build/obj-riscv32-nemu-interpreter/src/device/io/port-io.o -O2 -fPIE -lreadline -ldl -pie

\$

\$ █

NEMU代码构建

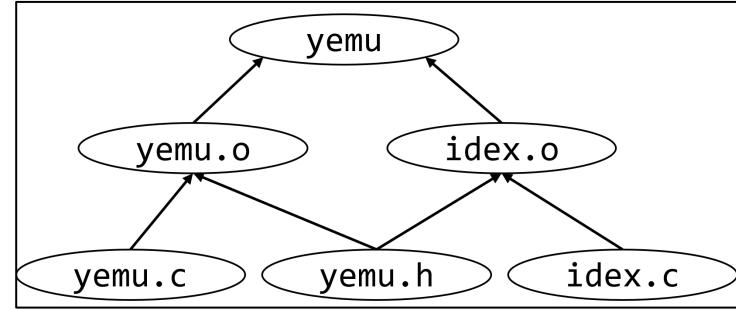
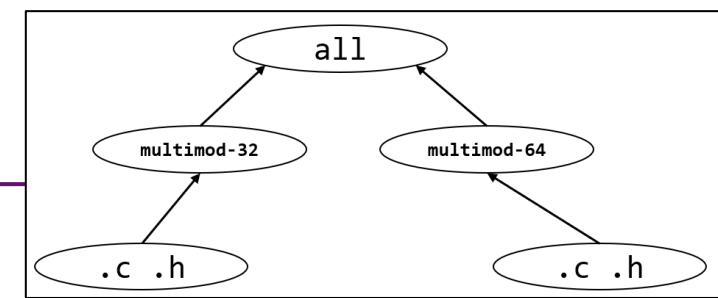
- Makefile 真复杂

- 放弃?

- 一个小诀窍

- 先观察 make 命令实际执行了什么 (trace)
 - RTFM/STFW: make 提供的两个有用的选项
 - -n 只打印命令不运行
 - -B 强制 make 所有目标

```
make -nB \
| grep -ve '^(\#|echo|mkdir)' \
| vim -
```



```

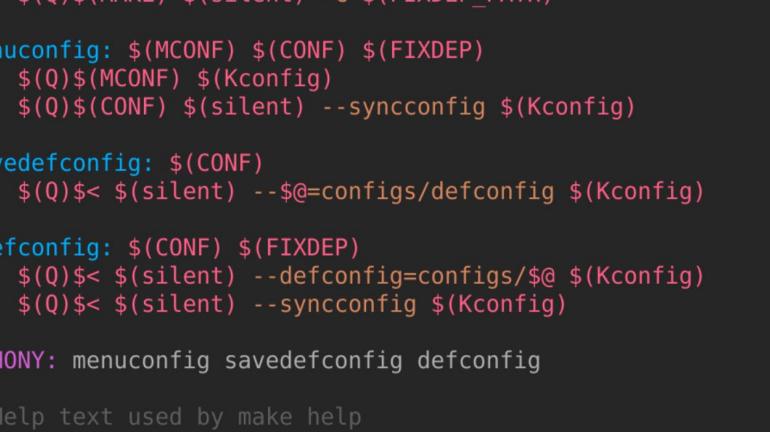
1 .DEFAULT_GOAL = app
2
3 # Add necessary options if the target is a shared library
4 ifdef SHARE
5 SO = -SO
6 CFLAGS += -fPIC
7 LDFLAGS += -rdynamic -shared -fPIC
8 endif
9
10 WORK_DIR = $(shell pwd)
11 BUILD_DIR = $(WORK_DIR)/build
12
13 INC_PATH := $(WORK_DIR)/include $(INC_PATH)
14 OBJ_DIR = $(BUILD_DIR)/obj-$(NAME)$(SO)
15 BINARY = $(BUILD_DIR)/$(NAME)$(SO)
16
17 CC ?= gcc
18
19 # Compilation flags
20 CC := $(CC)
21 LD := $(CC)
22 INCLUDES = $(addprefix -I, $(INC_PATH))
23 CFLAGS := -O2 -MMD -Wall -Werror $(INCLUDES) $(CFLAGS)
24 LDFLAGS := -O2 $(LDFLAGS)
25
26 OBJS = $(SRCS:.c=$(OBJ_DIR)/%.o)
27
28 # Compilation patterns
29 $(OBJ_DIR)/%.o: %.c
30 @echo + CC $<
31 @mkdir -p $(dir $@)
32 @$ (CC) $(CFLAGS) -c -o $@ $<
33 $(call call_fixdep, $(@:o=d), $@)
34
35 # Dependencies
36 -include $(OBJS:.o=.d)
37
38 # Some convenient rules
39
40 .PHONY: app clean
41
42 app: $(BINARY)
43
44 $(BINARY): $(OBJS) $(ARCHIVES)
45 @echo + LD $@
46 @$ (LD) -o $@ $(OBJS) $(LDFLAGS) $(ARCHIVES) $LIBS
47
48 clean:
49 -rm -rf $(BUILD_DIR)
~/Documents/ICS2021/ics2021/nemu/scripts/build.mk[2] [make] unix utf-8 Ln 46, Col 53/49
-- INSERT --
[0] 0:vim*

```

```
File Edit View Terminal Tabs Help Terminal - Makefile (~/ics2021/nemu) - VIM
19 include $(FILELIST_MK)
20
21 # Filter out directories and files in blacklist to obtain the fi
22 DIRS-BLACKLIST-y += $(DIRS-BLACKLIST)
23 SRCS-BLACKLIST-y += $(SRCS-BLACKLIST) $(shell find $(DIRS-BLACKL
24 SRCS-y += $(shell find $(DIRS-y) -name "*.c")
25 SRCS = $(filter-out $(SRCS-BLACKLIST-y),$(SRCS-y))
26
27 # Extract compiler and options from menuconfig
28 CC = $(call remove_quote,$(CONFIG_CC))
29 CFLAGS_BUILD += $(call remove_quote,$(CONFIG_CC_OPT))
30 CFLAGS_BUILD += $(if $(CONFIG_CC_LTO),-flto,)
31 CFLAGS_BUILD += $(if $(CONFIG_CC_DEBUG),-ggdb3, )
32 CFLAGS_BUILD += $(if $(CONFIG_CC_ASAN),-fsanitize=address, )
33 CFLAGS += $(CFLAGS_BUILD) -D_GUEST_ISA_=$(GUEST_ISA)
34 LDFLAGS += $(CFLAGS_BUILD)
35
36 # Include rules for menuconfig
```

Terminal - native.mk (~/ics2021/nemu/scripts) - VIM

```
1 include $(NEMU_HOME)/scripts/git.mk
2 include $(NEMU_HOME)/scripts/build.mk
3
4 include $(NEMU_HOME)/tools/difftest.mk
5
6 compile_git:
7     $(call git_commit, "compile")
8 $(BINARY): compile_git
9
10 # Some convenient rules
11
12 override ARGS ?= --log=$(BUILD_DIR)/ne
13 override ARGS += $(ARGS_DIFF)
14
15 # Command to execute NEMU
16 IMG ?=
17 NEMU_EXEC := $(BINARY) $(ARGS) $(IMG)
18
19 run-env: $(BINARY) $(DIFF_REF_SO)
20
21 run: run-env
22     $(call git_commit, "run")
23     $(NEMU_EXEC)
24
25 gdb: run-env
26     $(call git_commit, "gdb")
27     gdb -s $(BINARY) --args $(NEMU_EXEC)
scripts/native.mk
scripts/native.mk" 35L, 760C
```



The screenshot shows a terminal window with the title "Terminal - config.mk (~/ics2021/nemu/scripts) - VIM". The code displayed is a Makefile snippet for a configuration script. It includes definitions for menuconfig, savedefconfig, and defconfig, and lists menuconfig, savedefconfig, and defconfig as .PHONY targets. It also includes a help target with a descriptive comment.

```
File Edit View Terminal Tabs Help  
26 $(FIXDEP):  
27     $(Q)$(MAKE) $(silent) -C $(FIXDEP_PATH)  
28  
29 menuconfig: $(MCONF) $(CONF) $(FIXDEP)  
30     $(Q)$(MCONF) $(Kconfig)  
31     $(Q)$(CONF) $(silent) --syncconfig $(Kconfig)  
32  
33 savedefconfig: $(CONF)  
34     $(Q)$< $(silent) --$@=configs/defconfig $(Kconfig)  
35  
36 %defconfig: $(CONF) $(FIXDEP)  
37     $(Q)$< $(silent) --defconfig=configs/$@ $(Kconfig)  
38     $(Q)$< $(silent) --syncconfig $(Kconfig)  
39  
40 .PHONY: menuconfig savedefconfig defconfig  
41  
42 # Help text used by make help  
43 help:  
44     @echo ' menuconfig      - Update current config utilising a
```

```
File Edit View Terminal Tabs Help
17 CC ?= gcc
18
19 # Compilation flags
20 CC := $(CC)
21 LD := $(CC)
22 INCLUDES = $(addprefix -I, $(INC_PATH))
23 CFLAGS := -O2 -MMD -Wall -Werror $(INCLUDES) $(CFLAGS)
24 LDFLAGS := -O2 $(LDFLAGS)
25
26 OBJS = $(SRCS:.c=$(OBJ_DIR)/%.o)
27
28 # Compilation patterns
29 $(OBJ_DIR)/%.o: %.c
30     @echo + CC $<
31     @mkdir -p $(dir $@)
32     @$(CC) $(CFLAGS) -c -o $@ $<
33     $($call call_fixdep, $($@:o=.d), $@)
34
35 # Dependencies
36 -include $(OBJS:.o=.d)
37
38 # Some convenient rules
39
40 .PHONY: app clean
41
42 app: $(BINARY)
43
scripts/build.mk
```

NEMU代码构建

- 一个小诀窍
 - 先观察 make 命令实际执行了什么 (trace)
 - RTFM/STFW: make 提供的两个有用的选项
 - -n 只打印命令不运行
 - -B 强制 make 所有目标

```
make -nB \
| grep -ve '^(\#\|echo\|mkdir\)' \
| vim -
```

- 其实没有那么复杂
 - 就是一堆gcc -c (编译) 和一个gcc (链接)
 - 大部分Makefile都是编译选项
- Read the friendly source code!

AbstractMachine 代码构建

- 更长，更难读
 - 但我们在给大家留了一个小彩蛋

```
1 Makefile for AbstractMachine Kernels and Libraries
2
3 ### *Get a more readable version of this Makefile* by `make html` (requires python-markdown)
4 html:
5   cat Makefile | sed 's/^\\([^\#]\\)/ \\1/g' | markdown_py > Makefile.html
6 .PHONY: html
7
8 ## 1. Basic Setup and Checks
9
10 ### Default to create a bare-metal kernel image
11 ifeq ($(MAKECMDGOALS),)
12   MAKECMDGOALS = image
13   .DEFAULT_GOAL = image
14 endif
15
16 ### Override checks when `make clean/clean-all/html`
17 ifeq ($(findstring $(MAKECMDGOALS),clean|clean-all|html),)
18
19 ### Print build info message
20 $(info # Building $(NAME)-$(MAKECMDGOALS) [$(ARCH)])
21
```

###*Get a more readable version of this Makefile* by
`make html` (requires python-markdown)

html:

```
cat Makefile | sed 's/^\\([^\#]\\)/ \\1/g' | \
  markdown_py > Makefile.html
```

.PHONY: html

```
45
46 ## 2. General Compilation Targets
47
48 ### Create the destination directory (`build/$ARCH`)
49 WORK_DIR = $(shell pwd)
50 DST_DIR = $(WORK_DIR)/build/$(ARCH)
51 $(shell mkdir -p $(DST_DIR))
52
```

管理 控制 视图 热键 设备 帮助

```
$ ls  
am build klib LICENSE Makefile README scripts  
$ vim Makefile  
$ |
```

AbstractMachine 代码构建

- “现代”的文档编写方式
 - “docs as code”，例子： LLVM 使用 Doxygen [自动生成文档](#)

LLVM 14.0.0git

Main Page

Related Pages

Modules

Namespaces ▾

Classes ▾

Files ▾

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

- ▶ **N** `adjust`
- ▶ **N** `Allocaslices`
- ▶ **N** `false`
- ▶ **N** `INITIALIZE_PASS`
- ▶ **N** `llvm`
- ▶ **N** `std`
- ▶ **C** `_itt_api_info`
- ▶ **C** `_itt_api_info_20101001`
- ▶ **C** `_itt_global`
- ▶ **C** `_itt_group_list`
- ▶ **C** `_itt_thread_info`
- ▶ **C** `_JIT_Method_Id`
- ▶ **C** `_JIT_Method_Load`
- ▶ **C** `_JIT_Method_NIDS`
- ▶ **C** `_LineNumberInfo`
- ▶ **C** `AAAlignArgument`
- ▶ **C** `AAAaligncallsiteargument`
- ▶ **C** `AAAaligncallsitereturned`

TargetPassConfig

PC

Align attribute for functi

Align attribute deductio

LLVM 14.0.0git

Main Page Related Pages Modules Namespaces ▾ Classes ▾ Files ▾ Examples

Ilvm ▶ AA

Namespaces | Functions

Ilvm::AA Namespace Reference

Abstract Attribute helper functions. More...

Namespaces

PointerInfo

Functions

bool `isDynamicallyUnique (Attributor &A, const AbstractAttribute &QueryingAA, const Value &V)`
Return true if V is dynamically unique, that is, there are no two "instances" of V at runtime with different values. More...

bool `isValidInScope (const Value &V, const Function *Scope)`
Return true if V is a valid value in Scope, that is a constant or an instruction/argument of Scope. More...

bool `isValidAtPosition (const Value &V, const Instruction &Ctx, InformationCache &InfoCache)`
Return true if V is a valid value at position Ctx, that is a constant, an argument of the same function as Ctx, or an instruction in that function that dominates Ctx. More...

Value * `getWithType (Value &V, Type &Ty)`
Try to convert V to type Ty without introducing new instructions. More...

Optional< Value *> `combineOptionalValuesInAAValueLattice (const Optional< Value *> &A, const Optional< Value *> &B, Type *Ty)`
Return the combination of A and B such that the result is a possible value of both. More...

Constant * `getInitialValueForObj (Value &Obj, Type &Ty)`
Return the initial value of Obj with type Ty if that is a constant. More...

bool `getAssumedUnderlyingObjects (Attributor &A, const Value &Ptr, SmallVectorImpl< Value * > &Objects, const AbstractAttribute &QueryingAA, const Instruction *Ctx)`
Collect all potential underlying objects of Ptr at position Ctx in Objects. More...

bool `getPotentialCopiesOfStoredValue (Attributor &A, StoreInst &SI, SmallSetVector< Value * , 4 > &PotentialCopies, const AbstractAttribute &QueryingAA, bool &UsedAssumedInformation)`
Collect all potential values of the one stored by SI into PotentialCopies. More...

Detailed Description

Abstract Attribute helper functions.

Function Documentation

+ `combineOptionalValuesInAAValueLattice()`

Optional< Value *> Ilvm::AA::combineOptionalValuesInAAValueLattice (const Optional< Value *> & A,
const Optional< Value *> & B,
Type * Ty
)

Return the combination of A and B such that the result is a possible value of both.

B is potentially casted to match the type Ty or the type of A if Ty is null.

pydoc

- pydoc3 list

```
Help on class list in module builtins:

class list(object)
| list(iterable=(), /)

    Built-in mutable sequence.

    If no argument is given, the constructor creates a new empty list.
    The argument must be an iterable if specified.

Methods defined here:

__add__(self, value, /)
    Return self+value.

__contains__(self, key, /)
    Return key in self.

__delitem__(self, key, /)
    Delete self[key].

__eq__(self, value, /)
    Return self==value.

__ge__(self, value, /)
    Return self>=value.

__getattribute__(self, name, /)
    Return getattr(self, name).

__getitem__(...)
    x.__getitem__(y) <=> x[y]

__gt__(self, value, /)
    Return self>value.

__iadd__(self, value, /)
    Implement self+=value.

__imul__(self, value, /)
    Implement self*=value.

__init__(self, /, *args, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

__iter__(self, /)
    Implement iter(self).

__le__(self, value, /)
    Return self<=value.

:
```

总结

关于《计算机系统基础》习题课

- 教会大家 “计算机的正确打开方式”
 - 编程 ≠ 闷头写代码
 - 使用工具也是编程的一部分
 - version-control systems: git, svn, ...
 - build systems: make, cmake (C++), maven (Java), ...
 - shell: bash, zsh, ...
- 基本原则：任何感到不爽的事情都一定有工具能帮你
 - 如果真的没有，自己造一个的就会就来了
 - (不太可能是真的)
 - 但这将会是一份非常好的研究工作

End.

(RTFM & RTFSC)