

AOSCC 2026 | 欢迎

从0到w4*:

*w4为愿景，并非玩家实际舞萌分数

舞萌DX从入门到入土

CHIHUO & DeepSeek V4 Pro

舞萌DX玩家 & 人工智能助手



舞萌DX如何从入门到入土？

(思考中，用时114.51秒)

嗯，我们需要理解用户的问题：“舞萌DX如何从入门到入土？”这看起来像是一个玩笑式的提问，但实际上可能是在问舞萌DX（maimai DX）这款街机音乐游戏如何从新手入门到成为高手（或者说“入土”指深度沉迷或成为高手/老手，或者指彻底精通）。网络用语“从入门到入土”通常指从开始学习到非常精通、甚至沉迷至死的程度，是一种夸张说法。所以问题可能是：请给我一个从新手到高手的完整攻略/指南。

我需要确认舞萌DX的基本信息：这是SEGA开发的街机音乐游戏，有触摸屏和按钮操作。要给出一个详细的指南，涵盖入门基础、进阶技巧、高难度曲目挑战、成绩提升，以及可能涉及游戏文化、排位等。由于“入土”可以理解为成为顶端玩家，我需要覆盖从零基础到能打高难度谱面（比如MASTER/Re:MASTER 13+、14、15级）甚至达成理论值（AP/FC）的程度。同时要包含练习方法、设置调节、读谱技巧、体力分配、手部姿势、术语解释等。最终回答需要幽默风趣又实用，符合“从入门到入土”的梗。

这是一个关于街机音乐游戏“舞萌DX”（maimai DX）的入门问题。我需要用中文回答。

需要涵盖：舞萌DX是什么（简介），如何开始玩（注册、登录、基本操作），键位与音符介绍，难度选择与设置，基本技巧（如何打歌、判定、手部姿势），一些入门建议和常见术语，资源推荐（看谱面等）

答案结构：先定义舞萌DX，然后入门步骤：购币/开卡，登录，选歌，基本玩法，音符类型，判定，设置（速度、判定调整等），新手推荐曲目，礼仪，一些小贴士。要友好、易懂。确保不使用 markdown，因为要求是“你提供的信息不需要Markdown格式”，但是思考中可以用。最终回答需要纯文本。我可以使用自然换行。

现在组织回答。

我需要用流畅的中文回答，不要用Markdown。可以适当使用emoji但非必须。可能用户需要的就是一个简单的入门指南，所以要尽量实用。

最终输出纯文本。开始回答。

舞萌DX如何从入门到入土？

哈哈哈哈，你的观察极其敏锐！“从入门到入土”这个说法很幽默，那咱们就先稳稳地走进舞萌DX的大门，入土的事以后再说。

舞萌DX全名 maimai DX，外形像一台滚筒洗衣机，所以圈内俗称“洗衣机”。玩法是配合音乐，在恰当时机按下屏幕周围的8个按键，或触摸屏幕上的区域来不躲不藏不绕不逃稳稳地接住音符。

怎么开始玩？

找到机台，投币后一般会提示你登录。国内玩家通常绑定“舞萌DX”微信公众号，扫码登录，就能保存成绩和解锁内容。游客模式也行，没账号也可以直接玩，只是数据不保留。音符会从屏幕中心向外飞到周围的判定圈，你要在它们与圆圈重合时稳稳的接住它们，就可以拿高分啦。

一些礼貌和常识

不要大力拍打或滑动哟；旁边有人在玩时，不要急着投币挤人，等他打完后提示“插卡或投币”再操作；如果有人在排队，一般将卡或币放机台上代表轮候，店里规矩先观望一下就好；选歌时间有限，快速决定也是尊重后面玩家。

一句话总结：

用手和汗水就可以了。



AOSCC 2026 | 欢迎

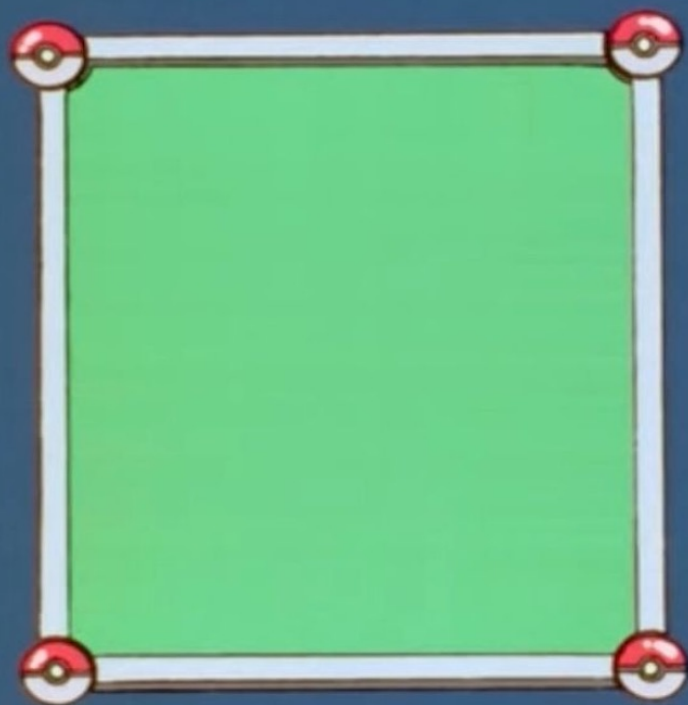
从水群到贡献者：

我的上下游维护踩坑手记

chihuo2104

社区贡献者





????
我是誰

ポケットモンスター
POCKET MONSTERS

“

我来这里讲什么？

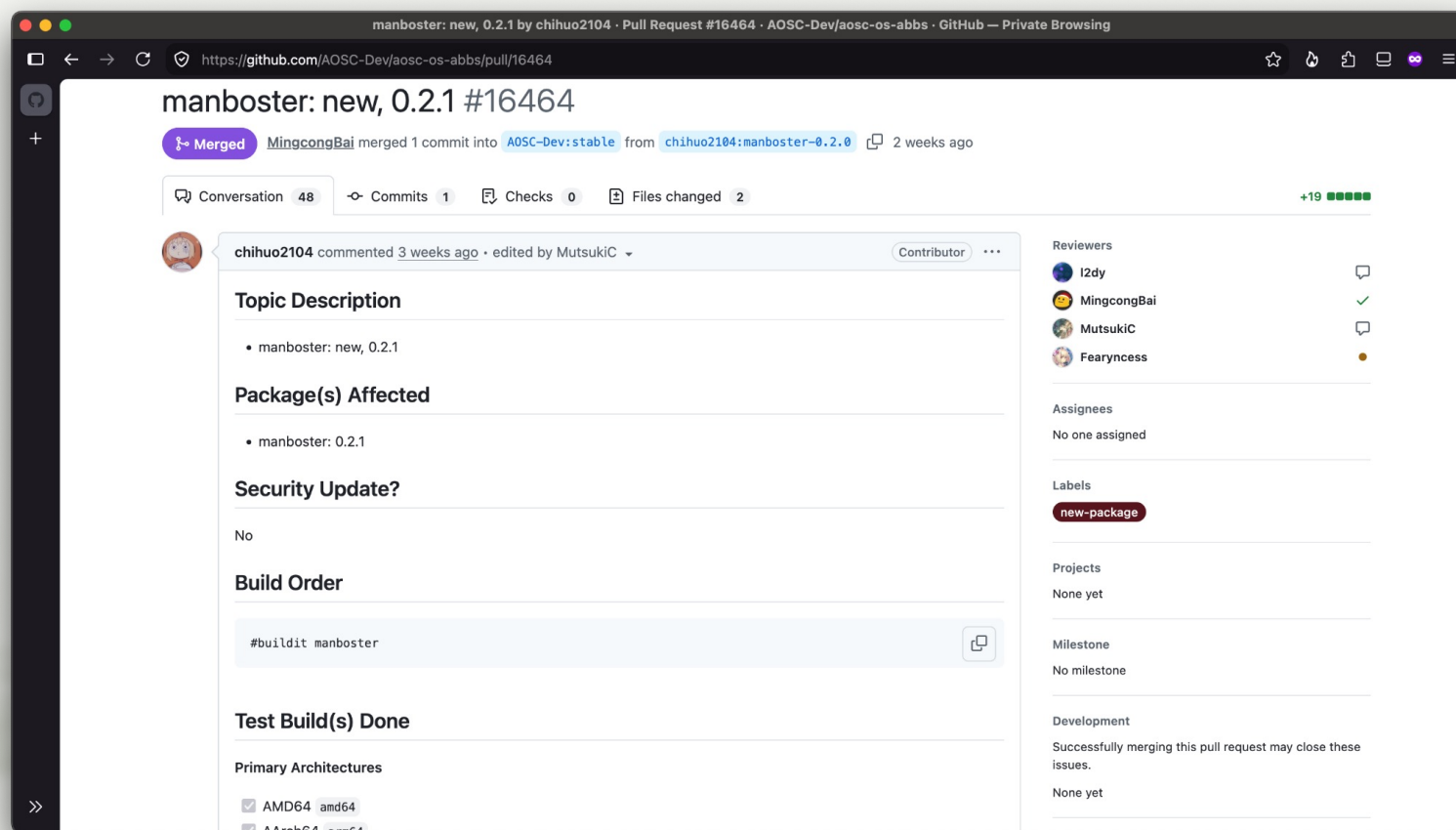
”



我打的 Manboster 是啥玩意？

- manboster.dev (Manbo Lobster, 曼波龙虾)
- 一个主打安全和轻量，面向个人的人工智能助理
- 使用 Golang 编写，日常占用 40MB
- 零信任架构，Hachimi 本地小模型判断工具调用是否合理
- 6.16 开源，6.22 进了 AOSC 的软件源
- 仍在实验性阶段，欢迎各位 oma 直接下载玩，有 bug 直接报给我
- 更多的就不说了，等会成产品发布会了...

#16464 的“毒打”



| #16464 的“毒打” 学到了什么？

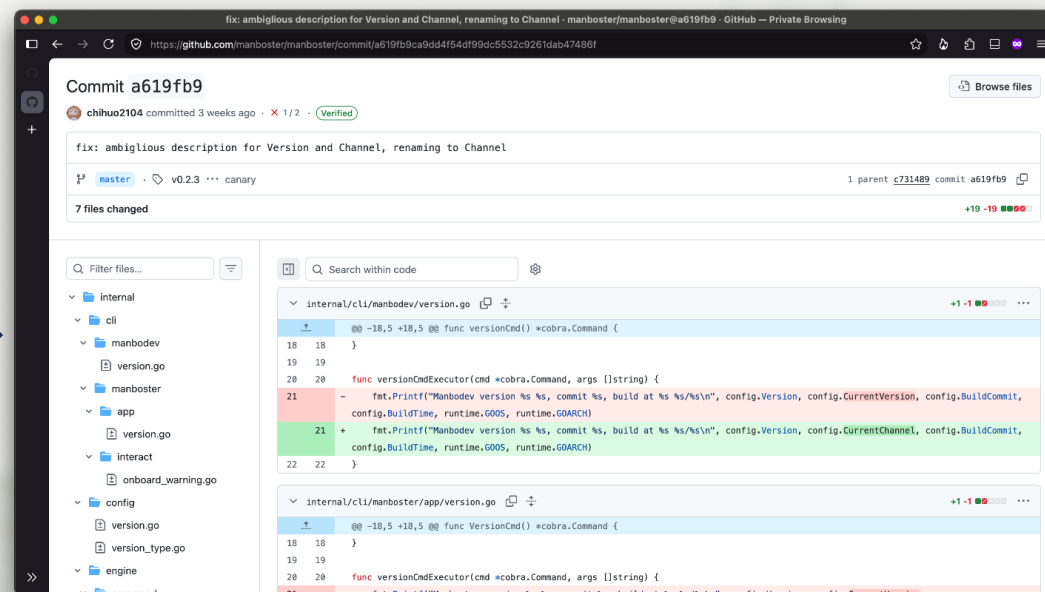
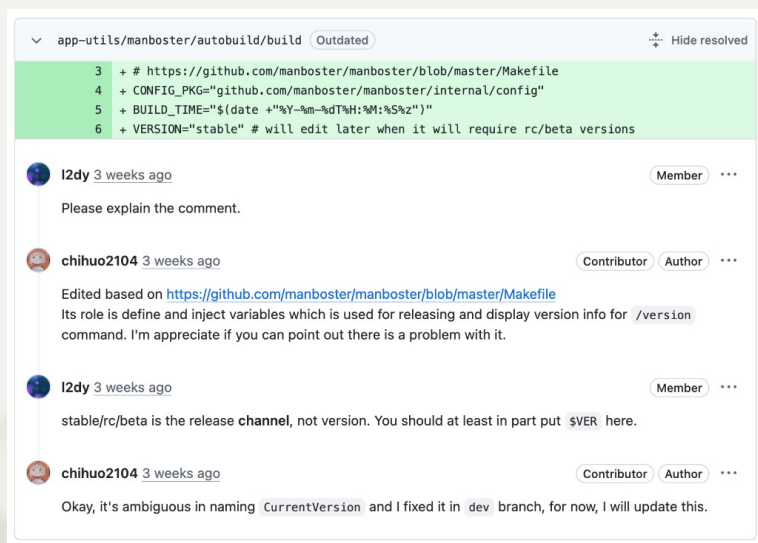
1. 下游不仅是消费者，也可以是贡献者



#16464 的“毒打”

1. 下游不仅是消费者，也可以是贡献者

Reviewer: @l2dy0 (Zero King)



下游提出在 Version / Channel 上表示不清

回上游，立即进行修复

#16464 的“毒打” 学到了什么？

1. 下游不仅是消费者，也可以是贡献者
2. 实践是检验“构建真理”的唯一标准



#16464 的“毒打”

2. 实践是检验“构建真理”的唯一标准

Reviewer: @fearyncess (网管)

Fearyncess commented on Jun 21

另外请在整改FAIL_ARCH之后，请求触发一下CI构建，test builds结果以CI为准。

ci如何触发？ FAIL_ARCH 是哪里有问题？还请指正。

图片

因为 yzma 依赖的purego的构建限制，暂无法支持全平台，只能支持 amd64 arm64 和 riscv64 这三个平台，后面版本会逐渐加上剩下平台的适配。

图片 图片

你现在gomod依赖的0.10.0是有除了mips64le loongson3以外支持的

那实际上是交叉编译没办法做编译，我没做过原生所以也不知道能不能正常编译成功，以及已经根据gomod的ABTYPE修改好了，现在重新push

交叉有问题是因为你go-sqlite版本老了， bump一下 (

“test builds结果以CI为准”

Reviewer: @MuTsukiC (tkm)

chihuo2104 on Jun 21

example:

```
aosc-os-abbs/app-shells/elvish/autobuild/defines
Line 4 in d945306
4 PKGDES="Statically linked interactive shell and scripting language"
```

Move these 2 definitions to define will break the app's versioning information if it is not injected into the ABTYPES gomod building shell. Is it injected into gomod ?

MutsukiC on Jun 21

Why not test by your own :)

chihuo2104 on Jun 21 • edited

Why not test by your own :)

You're right so I'm trying it

“Why not test by your own?”

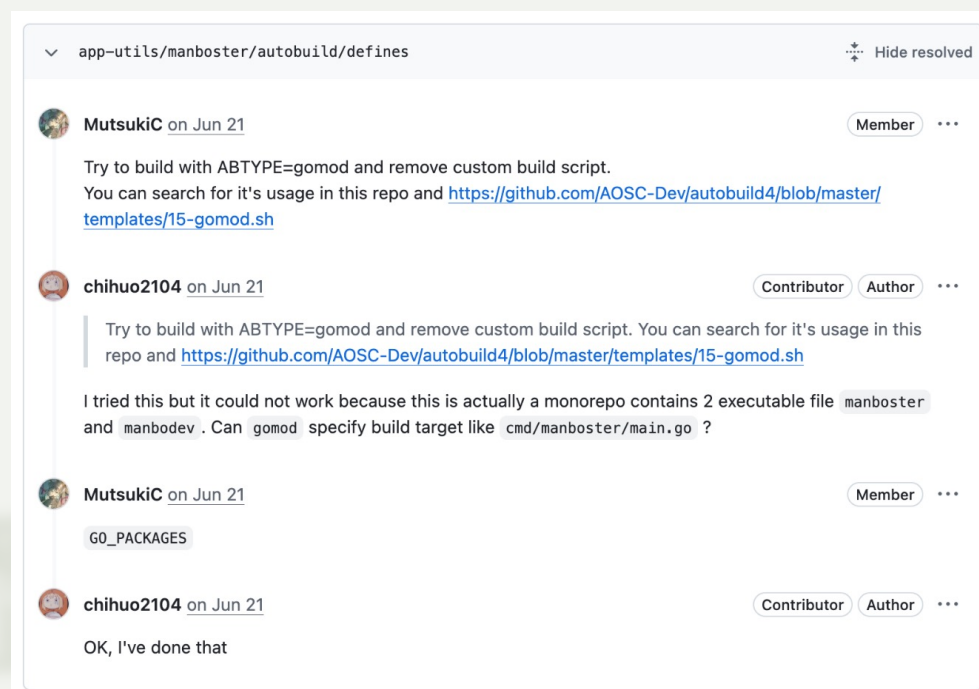
#16464 的“毒打” 学到了什么？

1. 下游不仅是消费者，也可以是贡献者
2. 实践是检验“构建真理”的唯一标准
3. 不会就去抄作业



#16464 的“毒打”

3. 不会就去抄作业



Reviewer: @MuTsukiC (tkm)

#16464 的“毒打”

3. 不会就去抄作业

app-utils/manboster/autobuild/defines

MutsukiC on Jun 21 Member ...

Try to build with ABTYPE=gomod and remove custom build script. You can search for it's usage in this repo and <https://github.com/AOSC-Dev/autobuild4/blob/master/templates/15-gomod.sh>

chihuo2104 on Jun 21 Contributor Author ...

Try to build with ABTYPE=gomod and remove custom build script. You can search for it's usage in this repo and <https://github.com/AOSC-Dev/autobuild4/blob/master/templates/15-gomod.sh>

I tried this but it could not work because this is actually a monorepo contains 2 executable file `manboster` and `manbodev`. Can `gomod` specify build target like `cmd/manboster/main.go` ?

MutsukiC on Jun 21 Member ...

G0_PACKAGES

chihuo2104 on Jun 21 Contributor Author ...

OK, I've done that

Reviewer: @MuTsukiC (tkm)

Autobuild4 - AOSC 文档

https://wiki.aosc.io/zh/developer/packaging/autobuild4/

搜索 AOSC Wiki

AOSC Wiki / 开发者 / 打包 / Autobuild4 / 其他语言: English

Autobuild4

本章内容:

软件包维护入门: 进阶教程 - AOSC 文档

https://wiki.aosc.io/zh/developer/packaging/advanced-techniques/

目前, 支持这些构建类型:

- self: 当存在 `autobuild/build` 文件时, 直接使用该文件作为构建脚本。
- autotools: 通常用于基于 GNU autotools 的源代码树, 在源根目录中具有可用的 `configure` 脚本, 或定义的 `$configure` 脚本。
- cmake: 用于基于 CMake 的源代码树, 生成并执行 Makefile, Autobuild3 在源代码树中 `CMakeList.txt`。
- cmakeninja: 与上面相同, 但生成并执行 Ninja 构建脚本。
- dummy: 生成空包, 通常只用于生成元包。
- dune: 用于基于 Dune 的源代码树 (通常用于 OCaml 源代码)。
- gomod: 用于使用了 Gomod 的 Go 语言源代码树。
- meson: 用于基于 Meson 的源代码树, 生成并执行 Ninja 构建脚本。
- npm: 用于 NPM 模块 (通常用于 Node.js 模块的源代码)。
- perl: 用于标准 CPAN 源代码树。
- plainmake: 用于包含已经写好的 Makefile 的源代码树, 因此可以使用 `make` 命令构建。
- python: 用于标准的 PyPI 源代码树。
- qtproj: 用于源代码树中包含 `.pro` 文件的 Qt 项目。
- ruby: 用于 RubyGems 源代码树。
- rust: 用于 Cargo 源代码树 (通常用于 Rust 源代码)。
- waf: 用于基于 Waf 的源代码树, Autobuild3 检测源代码树中的 `waf` 文件或脚本。

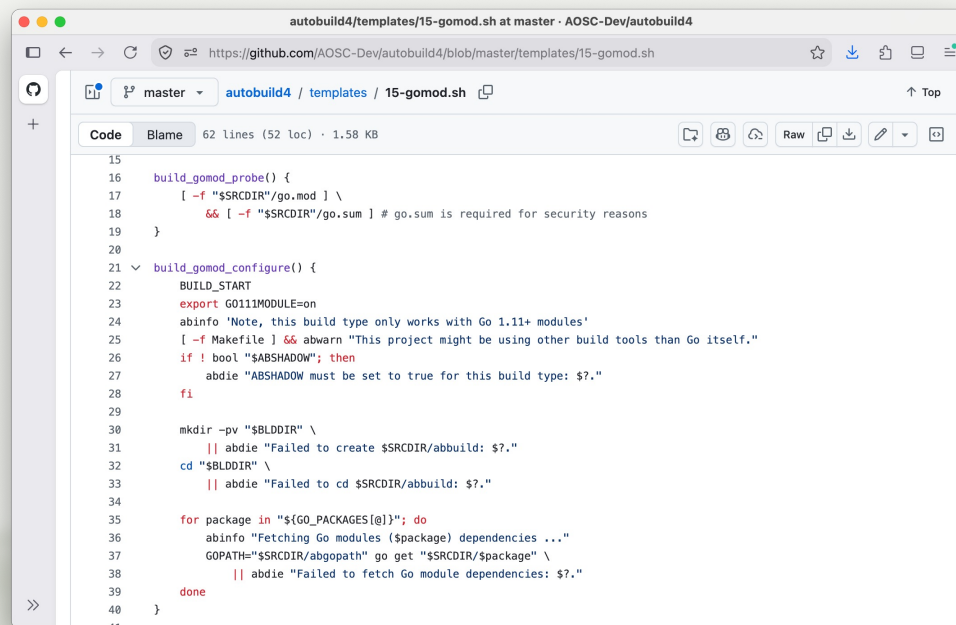
1.2 自定义构建系统/编译器参数

Autobuild3 集成了一系列最佳构建参数。但是, 有时这些参数与软件不完全兼容, 可能会引起问题。在这种情况下, 请查看 `Autobuild3` 的默认参数, 并根据需要覆盖对应的参数。完整的参数列表可以在 `Autobuild3 Wiki` 中找到。

你这作业让我怎么抄啊

#16464 的“毒打”

3. 不会就去抄作业

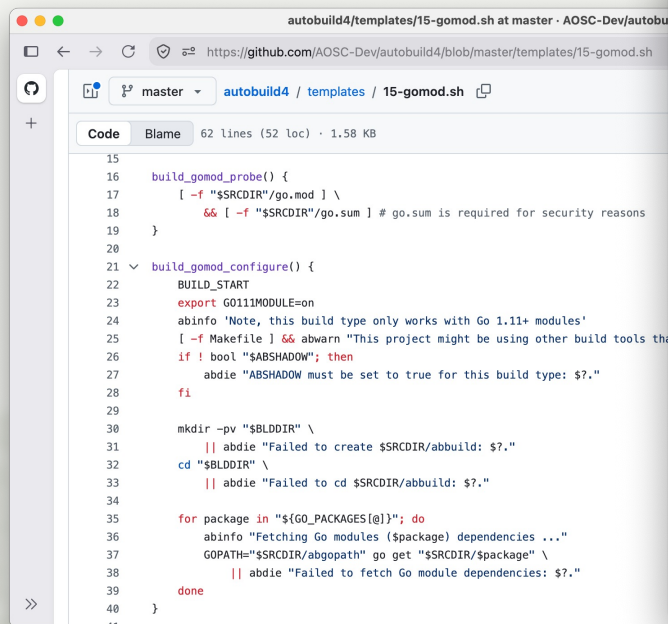


```
15
16 build_gomod_probe() {
17     [ -f "$SRCDIR"/go.mod ] \
18     && [ -f "$SRCDIR"/go.sum ] # go.sum is required for security reasons
19 }
20
21 build_gomod_configure() {
22     BUILD_START
23     export G011MODULE=on
24     abinfo 'Note, this build type only works with Go 1.11+ modules'
25     [ -f Makefile ] && abwarn "This project might be using other build tools than Go itself."
26     if ! bool "$ABSHADOW"; then
27         abdie "ABSHADOW must be set to true for this build type: $?"
28     fi
29
30     mkdir -pv "$BLDDIR" \
31     || abdie "Failed to create $SRCDIR/abuild: $?"
32     cd "$BLDDIR" \
33     || abdie "Failed to cd $SRCDIR/abuild: $?"
34
35     for package in "${G0_PACKAGES[@]}; do
36         abinfo "Fetching Go modules ($package) dependencies ..."
37         GOPATH="$SRCDIR/abgopath" go get "$SRCDIR/$package" \
38         || abdie "Failed to fetch Go module dependencies: $?"
39     done
40 }
41
```

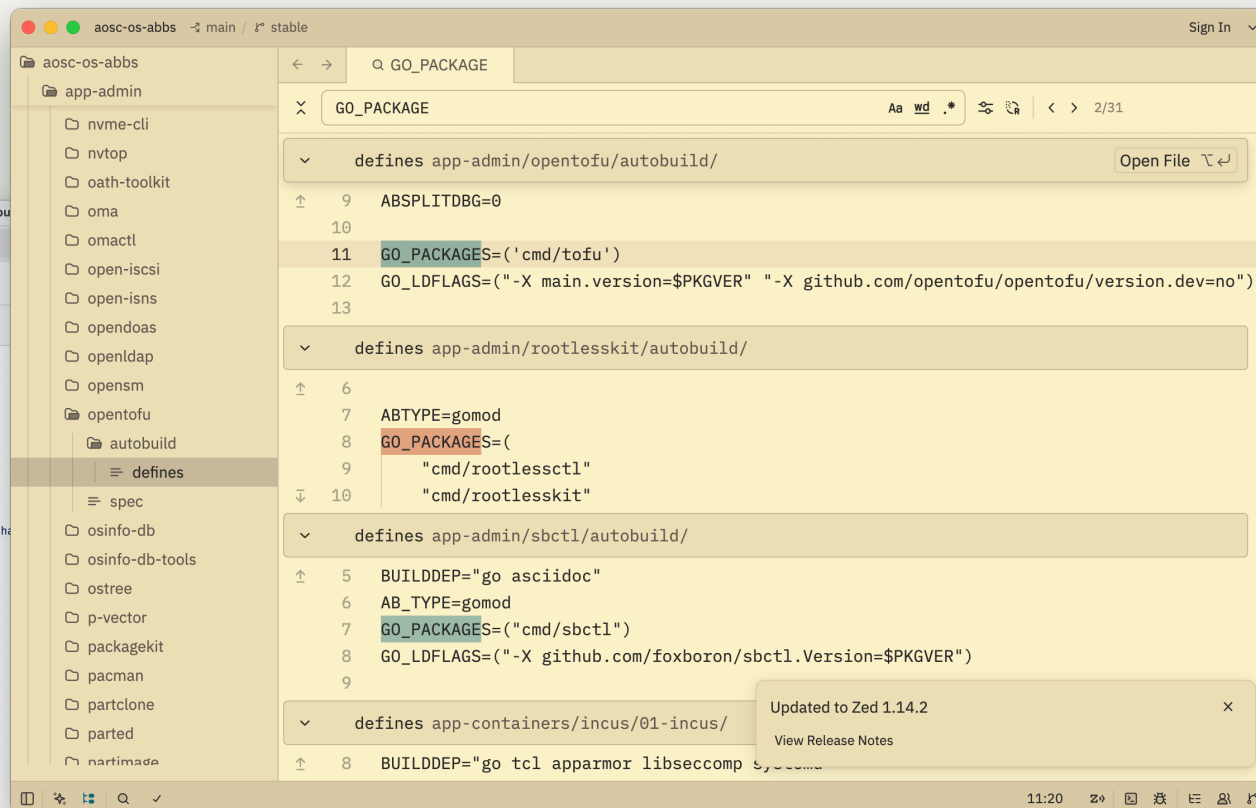
看不懂思密达

#16464 的“毒打”

3. 不会就去抄作业



The screenshot shows a GitHub web interface for the repository 'autobuild4/templates/15-gomod.sh'. The file is 62 lines long, 52 loc, and 1.58 KB. The code is a shell script for building Go modules. It includes functions like 'build_gomod_probe' and 'build_gomod_configure'. The script sets up a build environment with 'BUILD_START', 'export GO111MODULE=on', and 'abinfo'. It also includes a loop to fetch Go module dependencies for packages listed in 'GO_PACKAGES'.



The screenshot shows a code editor window with a file explorer on the left. The file explorer shows a directory structure for 'aosc-os-abbs' with subdirectories like 'app-admin', 'nvme-cli', 'nvtop', 'oath-toolkit', 'oma', 'omactl', 'open-iscsi', 'open-isns', 'opendoas', 'openldap', 'opensm', 'opentofu', 'autobuild', 'osinfo-db', 'osinfo-db-tools', 'ostree', 'p-vector', 'packagekit', 'pacman', 'partclone', 'parted', and 'nartimada'. The 'defines' section of the script is highlighted, showing the following code:

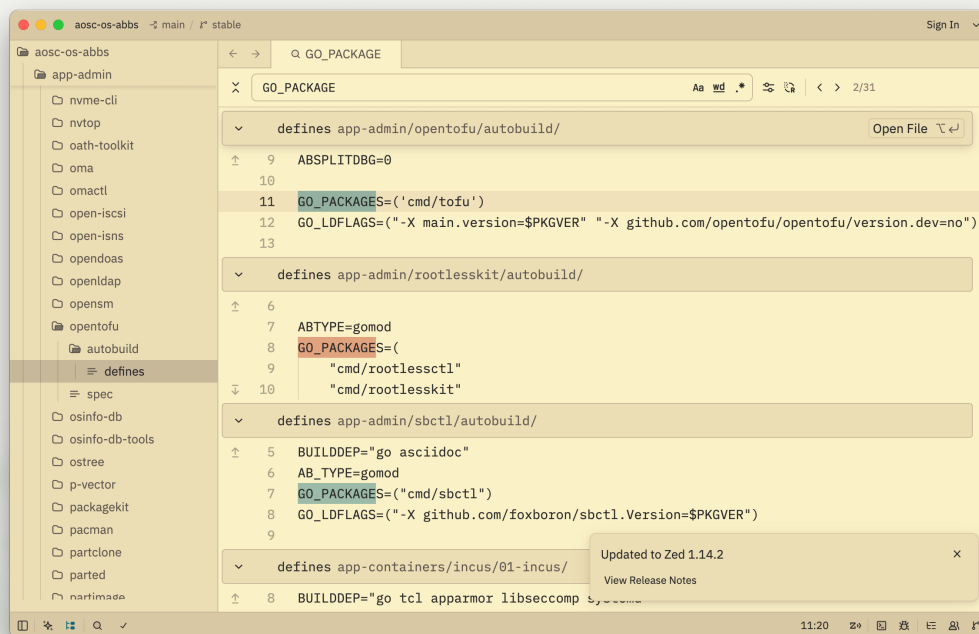
```
GO_PACKAGE
GO_PACKAGE
defines app-admin/opentofu/autobuild/
9 ABSPLITDBG=0
10
11 GO_PACKAGES=('cmd/tofu')
12 GO_LDFLAGS=(-X main.version=$PKGVER -X github.com/opentofu/opentofu/version.dev=no)
13
defines app-admin/rootlesskit/autobuild/
6
7 ABTYPE=gomod
8 GO_PACKAGES=(
9 "cmd/rootlessctl"
10 "cmd/rootlesskit"
)
defines app-admin/sbctl/autobuild/
5 BUILDDEP="go asciidoc"
6 AB_TYPE=gomod
7 GO_PACKAGES=("cmd/sbctl")
8 GO_LDFLAGS=(-X github.com/foxboron/sbctl.Version=$PKGVER)
9
defines app-containers/incus/01-incus/
8 BUILDDEP="go tcl apparmor libseccomp
```

看不懂思密达

终！于！找到了正确的写法！！！！

#16464 的“毒打”

3. 不会就去抄作业



```
GO_PACKAGE
defines app-admin/opentofu/autobuild/
9 ABSPLITDBG=0
10
11 GO_PACKAGES=('cmd/tofu')
12 GO_LDFLAGS=(-X main.version=$PKGVER" "-X github.com/opentofu/opentofu/version.dev=no")
13

defines app-admin/rootlesskit/autobuild/
6
7 ABTYPE=gomod
8 GO_PACKAGES=(
9 "cmd/rootlessctl"
10 "cmd/rootlesskit"

defines app-admin/sbctl/autobuild/
5 BUILDDEP="go asciidoc"
6 AB_TYPE=gomod
7 GO_PACKAGES=("cmd/sbctl")
8 GO_LDFLAGS=(-X github.com/foxboron/sbctl.Version=$PKGVER")
9

defines app-containers/incus/01-incus/
8 BUILDDEP="go tcl apparmor libseccomp"
```

终！于！找到了正确的写法！！！！

用仓库包抄作业没啥问题
But
文档去哪了？

#16464 的“毒打” 学到了什么？

1. 下游不仅是消费者，也可以是贡献者
2. 实践是检验“构建真理”的唯一标准
3. 不会就去抄作业
4. 注释不是写给现在的自己看的

#16464 的“毒打”

4. 注释不是写给现在的自己看的

Reviewer: @MingcongBai (特首)



MingcongBai added the **new-package** label on Jun 22

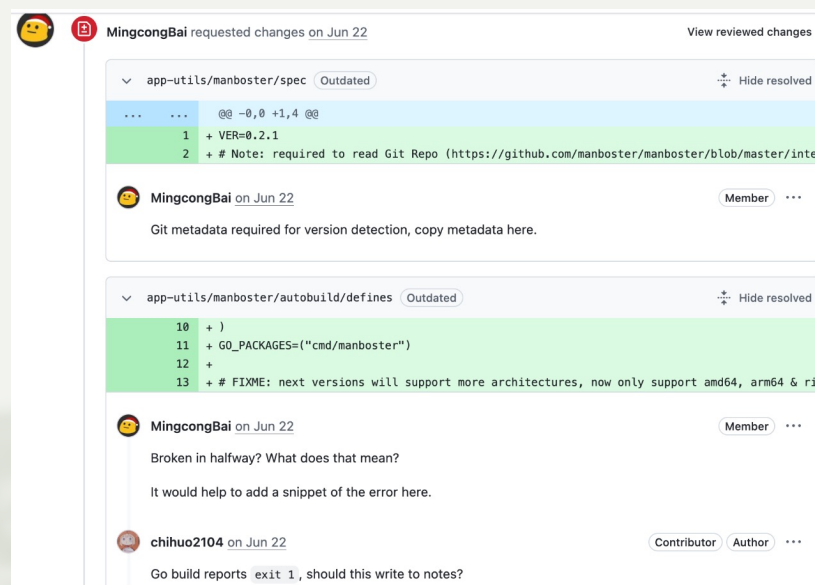
结束了吗?

No.

#16464 的“毒打”

4. 注释不是写给现在的自己看的

Reviewer: @MingcongBai (特首)



指出模棱两可的注释和错误提示

#16464 的“毒打” 学到了什么？

1. 下游不仅是消费者，也可以是贡献者
2. 实践是检验“构建真理”的唯一标准
3. 不会就去抄作业
4. 注释不是写给现在的自己看的

| #16464 的“毒打”

学到了什么？

1. 下游不仅是消费者，也可以是贡献者
2. 实践是检验“构建真理”的唯一标准
3. 不会就去抄作业
4. 注释不是写给现在的自己看的

有什么收获？

下游有些时候是上游最严厉的测试和CR
不要没有测就说不能用，跑了才知道

虽然代码也是文档，但是文档们都去哪了
注释是写给未来的维护者(或者你)看的

#16464 的“毒打”

<https://github.com/AOSC-Dev/aosc-os-abbs/compare/5ad29a8bcbe43fe7a940ab6c376829fa00ca362c..54a027f0109da449a1f0ce744b41f7ceea193c63>

force-pushed ^ v Highlight All Match Case Match Diacritics Whole Words 1 of 10 matches



✓ MingcongBai approved these changes on Jun 22



MingcongBai merged commit 7af6cda into AOSC-Dev:stable on Jun 22



MutsukiC requested changes on Jun 21

```
app-utils/manboster/autobuild/defines Outdated
5 + PKGDES="AI agent built for computer-use"
6 +
7 + ABTYPE="gomod"
8 + GO_LDFLAGS=(-X '$CONFIG_PKG.CurrentVersion=
```

MutsukiC on Jun 21

```
GO_LDFLAGS=(
  "-X 'github.com/manboster/manboster/internal/con
```

You don't have CONFIG_PKG defined

chihuo2104 on Jun 22

Fixed



MutsukiC commented on Jun 22

```
tsuki@DESKTOP-63F01EJ [ ~ ] $ manboster version
Manboster version 0.2.1 stable, commit 2328cf, build at 202
tsuki@DESKTOP-63F01EJ [ ~ ] $ manboster
Welcome to Manboster!
```

Mutsuki

尾巴霍霍 | HuoSeek-R1 Powered?

然后扔了个change request到现在还没有声音

昨晚困死了脑子不太好使 commit=tags/v\${VER}::copy-repo=true
中间那俩冒号应该是一个分号

09:29

Mutsuki

尾巴霍霍 | HuoSeek-R1 Powered?

所以实际上是长啥样的

```
shell
git::commit=tags/v${VER};copy-repo=true::repo}
```

09:30

有空改一下我排个构建

2 09:31

日了狗

Forwarded from Mutsuki

@purole 方便的话跟霍霍说下 GO_PACKAGES 只有一个的话不用换行 (00:13)

Forwarded from Mutsuki

然后 srcs 改成 git::commit=tags/v\${VER}::copy-repo=true::https://github.com/manboster/manboster (1 00:13)



尾巴霍霍 00:13

不觉得这很酷吗



Mutsuki

测试构建排了

22/6/26



Mutsuki

其实卡的没那么死

22/6/26



Mutsuki

不用

22/6/26

Mutsuki

msg + title

22/6/26

Mutsuki

直接改就行

22/6/26

Mutsuki

另外你直接上 0.2.1 吧-省的再...

22/6/26

Mutsuki

另外一般来说 PKGNAME 不加...

22/6/26

Mutsuki

不急

22/6/26

Mutsuki

有空改一下我排个构建

22/6/26

Mutsuki

git::commit=tags/v\${VER}::copy-repo=true::https://github.com/manboster/manboster

22/6/26

Mutsuki

昨晚困死了脑子不太好使

22/6/26

#16464 的“毒打”

TL;DR一下

- reviewers 对代码、构建和注释的严谨、细致入微的指导和细节的死磕
- 协作过程中有的一些不可避免的冲突和沟通问题，保持耐心
- **打一个包就像生了一个孩子，你不能只管生不管养**
- 维护下游、打patch、听用户反馈、给上游 contribute/反馈沟通 etc... 也是维护者的活
- 对社区的日常运作有了真实的一瞥——原来 ta 们究竟是在构建一些什么东西
- 结果，我的包进源才三个小时……就……（哽咽）

打完了，进库了，之后呢？

结果，我的包进源才三个小时……就……（哽咽）

19:28

- Manboster 进源

22:40

- 群友报告
Onboarding P0级
错误

22:41

- 排查问题，修复，
发版

23:11

- 0.2.2 发布

23:45

- 残留bug发现

次日0:22

- 0.2.3 发布 merge
发新版本

| 打完了，进库了，之后呢？

如果我不是 Manboster 的上游开发者呢？



打完了，进库了，之后呢？

如果我不是 Manboster 的上游开发者呢？

19:28

- Manboster 进源

22:40

- 群友报告
Onboarding P0级错误

22:41

- 我去 Manboster 仓库开issue

次日 13:25

- 开发者 chihuo2105
回复我说没问题 下个版本修

第三天 15:14

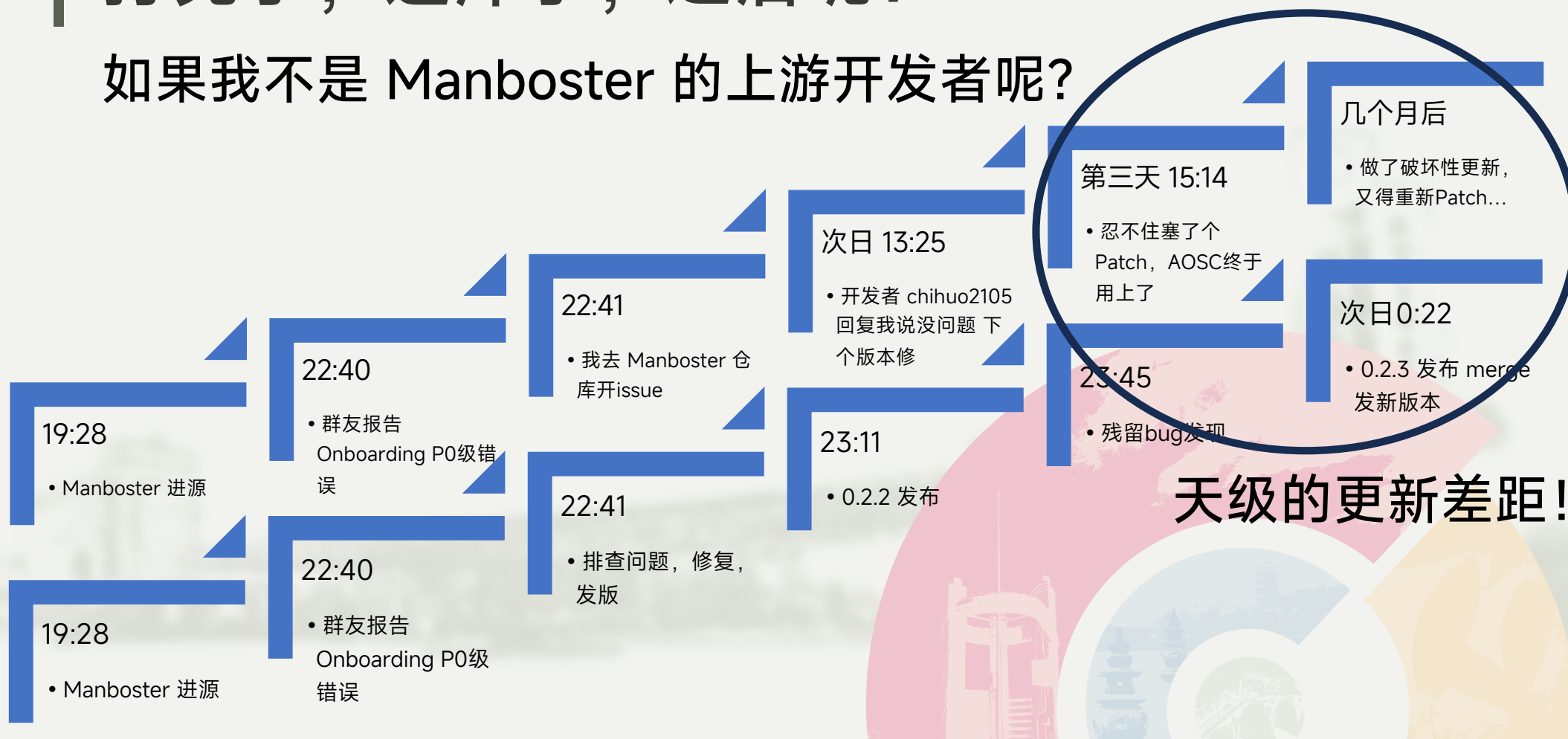
- 忍不住塞了个
Patch, AOSC终于
用上了

几个月后

- 做了破坏性更新，
又得重新Patch...

打完了，进库了，之后呢？

如果我不是 Manboster 的上游开发者呢？



打完了，进库了，之后呢？

上下游一体的 Pros & Cons

Pros

1. 上下游沟通链路极短—~~没有~~中间商赚差价
2. 由世界上最懂这个软件的人维护和处理问题，所以定位和修复速度可以很快
3. 下游环境本身又成为上游新的测试场景，可以持续获得真实用户反馈
4. ~~理论上~~可以在清凉的夏夜进行无忧的安眠

Cons

1. 要同时理解上游开发和发行版规范，上下游两套上下文切换的心智成本很高
2. 我被车撞了谁来接手我的包呢
3. ~~我成男妈妈了，那我缺的时间这一块谁给我补啊~~—(otto音

想说的一些话

- 对想尝试打包的朋友们：别怕开PR，开就行了；别怕Review的Comment多，改就行了！遇到问题不要慌，多学；遇到分歧保持耐心，搞明白所以然
- 对上游开发者们：“works on my machine” != 发行版能打包
稳定的构建方式、清楚的依赖说明、准确的版本和变更信息都会让下游少受很多苦；如果有机会的话，自己试着打一次包会更理解下游到底在干什么。
- 对 AOSC社区：Autobuild4 很好用，但新人不应该必须靠 reviewer 和“抄作业”知道它怎么用，我的建议还是完善 Autobuild4 的文档。
如果可以的话，这个活能不能让我来做？

“

感谢聆听！

”

有想法欢迎线下堵人或联系 i@chihuo2104.dev

特别感谢：AOSC社区、杭电ICU社团、审稿志愿者们、藿林郭勒、Aris(@Alice_8585) 和 Muji(@MakiTogawa)
素材来源：AOSC社区、Microsoft Office、MiSans、GitHub Pull Request(AOSC-Dev/aosc-os-abbs#16464)