

# RUCE 中文音库制作指南

适用版本：RUCE1.1.0-alpha3 或与 alpha3 兼容的版本

语言：汉语普通话

操作系统：本教程面向 Windows 用户，但大部分过程同时适用于 Linux 和 OSX。

本教程包括如何从设计、录制开始，直到制作出 RUCE 音库的过程。

理论上该过程同样适用于日语及其它类似语言，但没有经过测试，不保证质量。

# 准备工作

要制作一个 RUCE 中文音库你需要：

- 一台电脑和 Audacity 软件 (<http://audacity.sourceforge.net>)
- RUCE 和 Rocaloid 音库制作工具链 (与 RUCE 一同发布)
- 基于 MSYS<sup>1</sup> 的音库开发环境 ([www.rocaloid.org](http://www.rocaloid.org))
- 录音和放音设备
- 耐心和对完美的追求

<sup>1</sup> MSYS 是一个在 Windows 中模拟出的小型 Unix 环境。

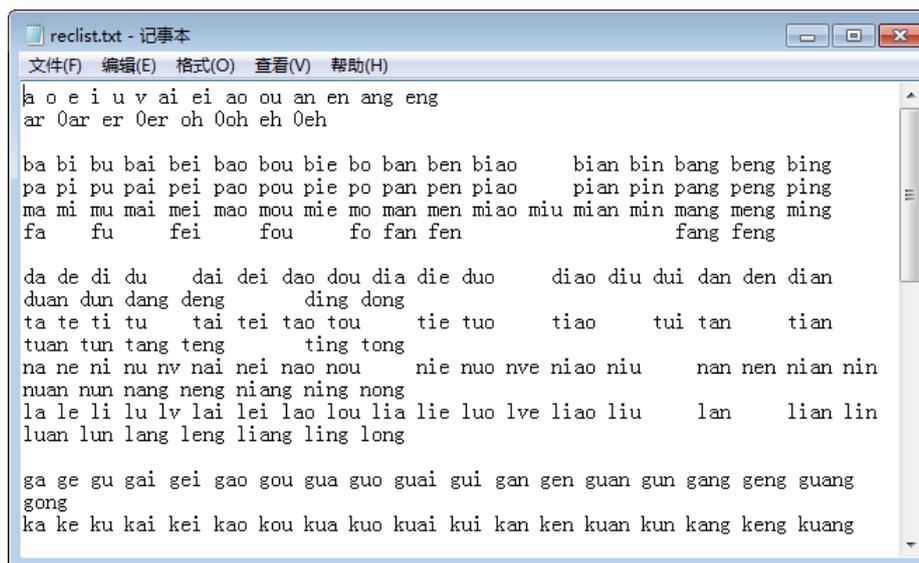
# 设计录音表

- 录音表是一个文本文档，包含了音库中所有发音的拼音。
- 可供参考的录音表 (by farter)：

[http://blog.sina.com.cn/s/blog\\_711e86460101jt89.html](http://blog.sina.com.cn/s/blog_711e86460101jt89.html)

其中 ' 号应替换成 . ，防止一些不兼容问题的发生。

- 存储为 reclist.txt ，可以有空格、 Tab 、或换行，例如下图：



```
reclist.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
a o e i u v ai ei ao ou an en ang eng
ar 0ar er 0er oh 0oh eh 0eh

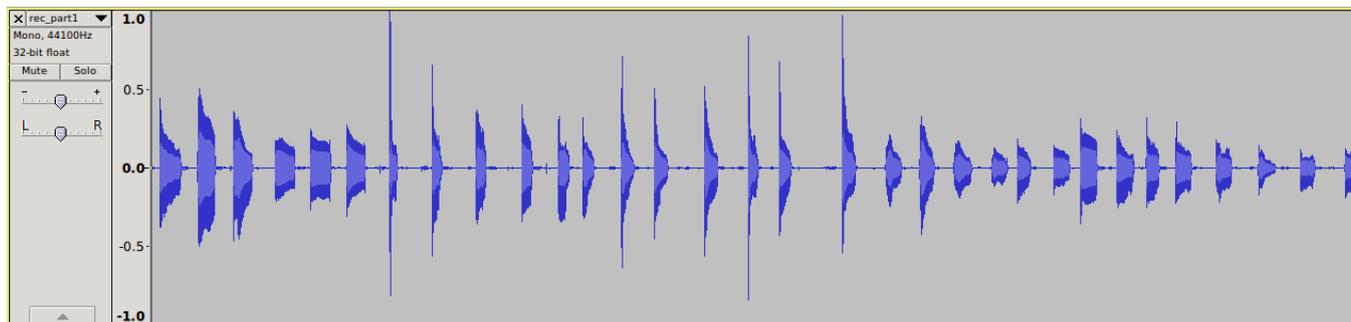
ba bi bu bai bei bao bou bie bo ban ben biao      bian bin bang beng bing
pa pi pu pai pei pao pou pie po pan pen piao      pian pin pang peng ping
ma mi mu mai mei mao mou mie mo man men miao miu mian min mang meng ming
fa  fu      fei      fou      fo fan fen                      fang feng

da de di du      dai dei dao dou dia die duo      diao diu dui dan den dian
duan dun dang deng      ding dong
ta te ti tu      tai tei tao tou      tie tuo      tiao      tui tan      tian
tuan tun tang teng      ting tong
na ne ni nu nv nai nei nao nou      nie nuo nve niao niu      nan nen nian nin
nuan nun nang neng niang ning nong
la le li lu lv lai lei lao lou lia lie luo lve liao liu      lan      lian lin
luan lun lang leng liang ling long

ga ge gu gai gei gao gou gua guo guai gui gan gen guan gun gang geng guang
gong
ka ke ku kai kei kao kou kua kuo kuai kui kan ken kuan kun kang keng kuang
```

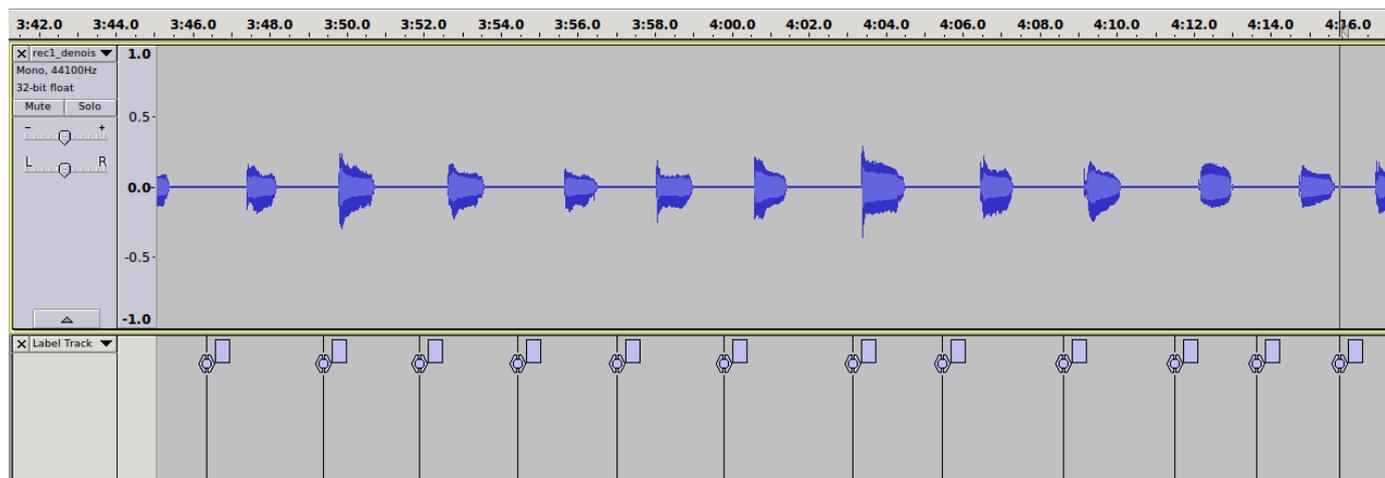
# 录音

- 按照录音表的顺序，从左到右从上到下，依次朗读录音表中的拼音。
- 要求吐字清晰，尽量保持音调不变。单个拼音时长 0.5s 以上。
- 格式：单声道，16bit，采样率 32000Hz 至 48000Hz 均可。  
( 若需要兼容 UTAU，采样率必须是 44100Hz )
- 不推荐用手机或电脑内置麦克风录制（主要问题是噪音和某些频段被削弱的现象）。推荐的设备是录音笔，或入门级的声卡和麦克风即可。



# 处理和分割

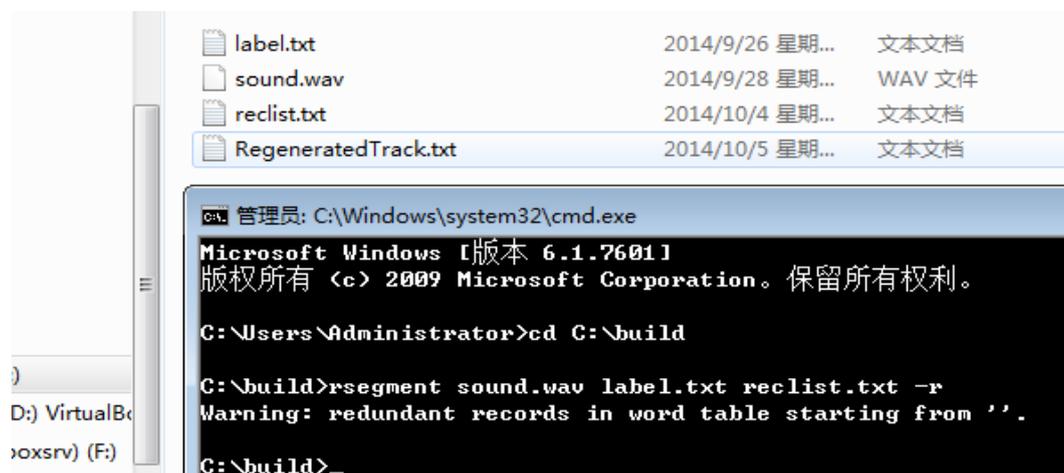
- 对原始录音进行适度的降噪，并且确保相邻两个发音之间为静音。
- 将处理后的录音文件导入 Audacity，使用 Ctrl+B 快捷键进行标注：



- 您不需要自行分割录音文件，这将在稍后由工具自动完成。

# 处理和分割

- 标注完成后，File → Export Labels... 导出标签轨到 label.txt。
- 请核对标签是否和录音表一致。
- 一个方法是，启动命令提示符，进入存放相关文件目录，输入  
rsegment 录音音频文件 标签文件 录音表文件 -r
- 将会生成 RegeneratedTrack.txt，将此文件导入 Audacity 检查标注是否一致。



# 编写音库源代码

- 从 RUCE alpha3 开始，音库的制作不再是手动调用工具处理各种文件，而是类似软件开发，通过 Makefile 代码和数据文件生成音库。
- 尽管在软件的构建与开发中，类似的流程已经相当常见，但是这种方式在音库的制作中仍然比较新颖。接下来的几页会带给您一个概念，再详细讲解具体的操作方式。
- 如果您觉得内容枯燥，不妨先跳过，按照指示完成一个初步的音库，再回过头来继续阅读。
- 接下来的流程将全部在 MSYS 环境中完成，这是为了保证音库能在所有操作系统下顺利构建。

# 流程

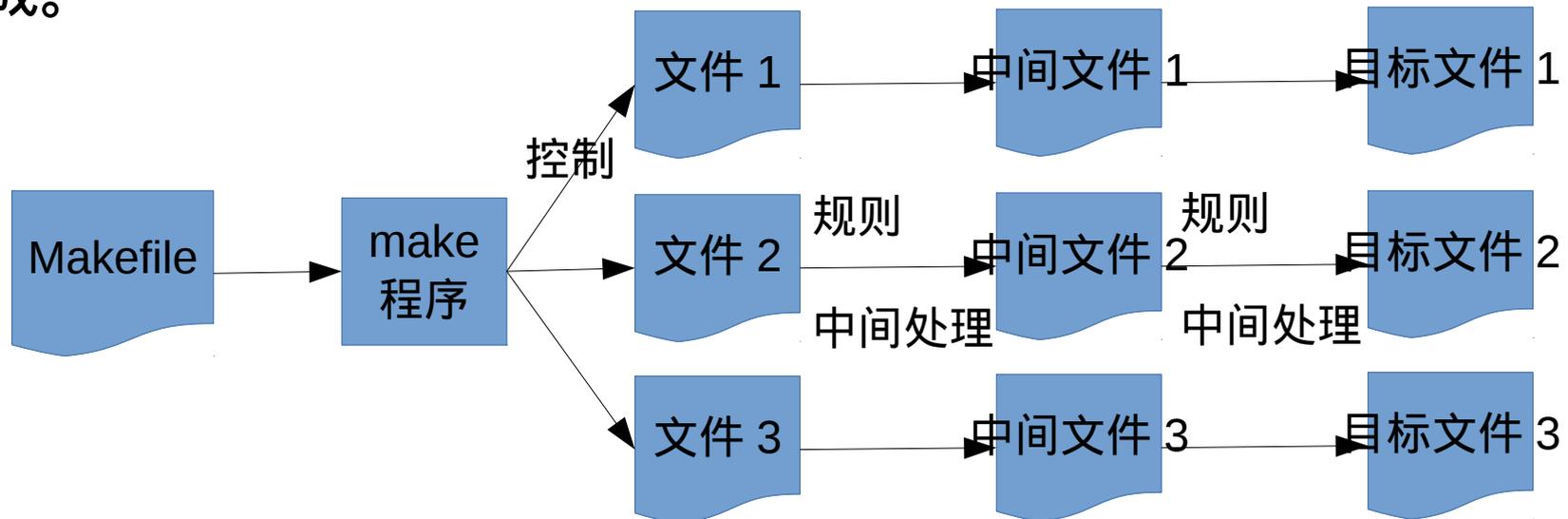
为 RUCE alpha3 及后续兼容版本制作音库的大体流程如下

- 根据录音表和标签轨文件分割录音文件
- 对分割出来的 wav 文件进行前期处理，例如切掉静音、归一化音量
- 分析 wav 文件，生成 rudb 文件
- 对 rudb 文件进行后期处理，修补瑕疵

Rocaloid 工具链包含了完成上述操作的所有工具。 Makefile 所要做的是记录这些操作的命令。

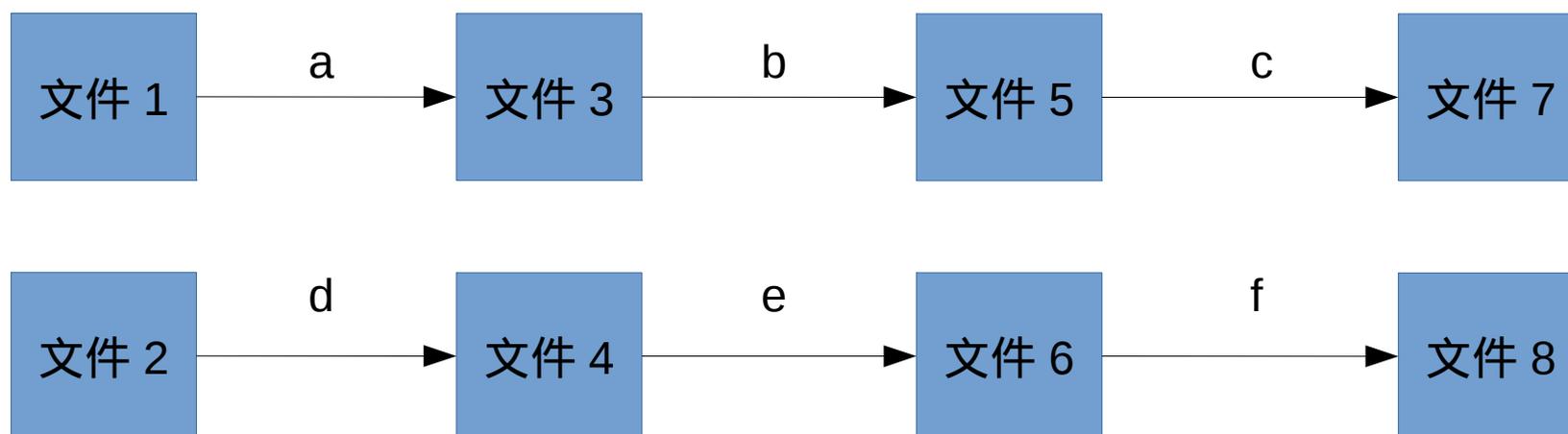
# Makefile 是什么？

- 无论软件还是音库的开发中，对于若干文件，都有相同或不同的操作流程，有时一个文件的生成需要若干步骤。
- Makefile 是一个文本文件，它记录了各个文件及临时生成的文件之间的依赖关系，以及如何一步步转换。
- 调用 make 程序，它会自动读取 Makefile 并找出文件的依赖关系，按照 Makefile 中预先写好的规则自动对文件进行处理、生成。



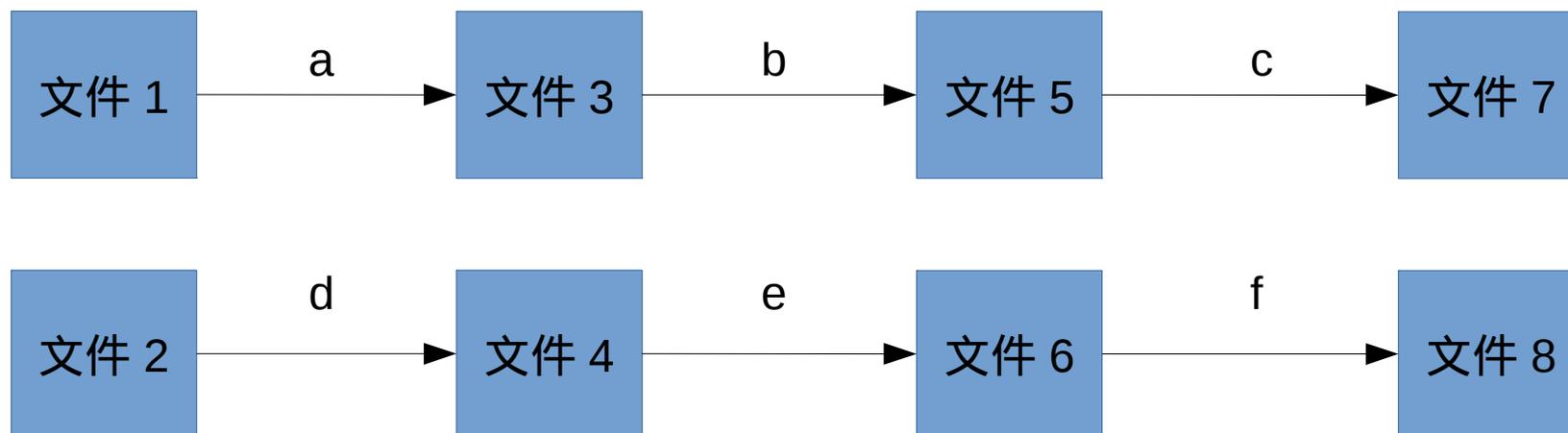
# 使用 Makefile 有什么优势？

- 假如我们要处理 2 个音库文件，每个文件需要 3 个步骤，每个步骤都会生成一个新的文件，最终生成我们需要的文件 7 和文件 8：



- 使用传统的音库制作方法需要 6 次操作。如果当操作完成后，我们发现这两个文件存在问题，而问题出在步骤 a、d 上，那么我们需要把 a 和 d 重新进行一遍，然而这会修改文件 3 和文件 4，因此我们又需要重新进行步骤 b、e，以此类推把 6 次操作全部重新进行一遍。

# 使用 Makefile 有什么优势？



- 如果用 Makefile 记录下 6 次操作，那么只需修改 Makefile 中对于 a、d 步骤的记录，可由程序自动执行之后的步骤，大大减少了人工劳动。
- 可以直接发布音库的 Makefile 和数据文件，而不需发布最终完成的文件，这样将文件体积缩小了一个数量级。
- 多人协作开发音库时，方便数据共享和版本控制。

# 从模板开始

- 我们了解编写 Makefile 具有一定技术门槛，因此提供了 Makefile 模板。为了让 RUCE 音源尽量统一，我们希望所有音源都能参照此模板编写。
- 模板下载地址：  
<https://github.com/Rocaloid/Soundbank-Template/archive/master.zip>
- 此模板包含多个文件：  
Makefile：主 Makefile  
src/PitchModel.json：音高变化补充参数  
src/postmakes.make：指定后期 rudb 文件处理规则的 Makefile  
src/premakes.make：指定前期 wav 文件处理规则的 Makefile

# 从模板开始

- 在音库开发环境的 home 目录下创建一个子目录，将模板的所有文件和目录拷贝到其下。将录音表文件 reclist.txt 和 Audacity 标签轨文件 label.txt 拷贝到 src 目录下。
- 录音文件可以是 wav ， flac ， 或 ogg 格式。但不支持 mp3 <sup>1</sup>。
- 编辑 Makefile 文件 ， 修改 sound.wav : | src/sound.flac 这行附近的代码 <sup>2</sup> ：

```
34 sound.wav : | src/sound.flac|
35     sox src/sound.flac sound.wav
36     rsegment sound.wav src/label.txt src/reclist.txt
37     @ mv sound.wav sound.wxx
38     @ for i in *.wav; do \
39         mv $$i $$i'1'; \
40     done
41     @ mv sound.wxx sound.wav
```

- sound.flac 应被修改为 src 目录下的录音文件名。

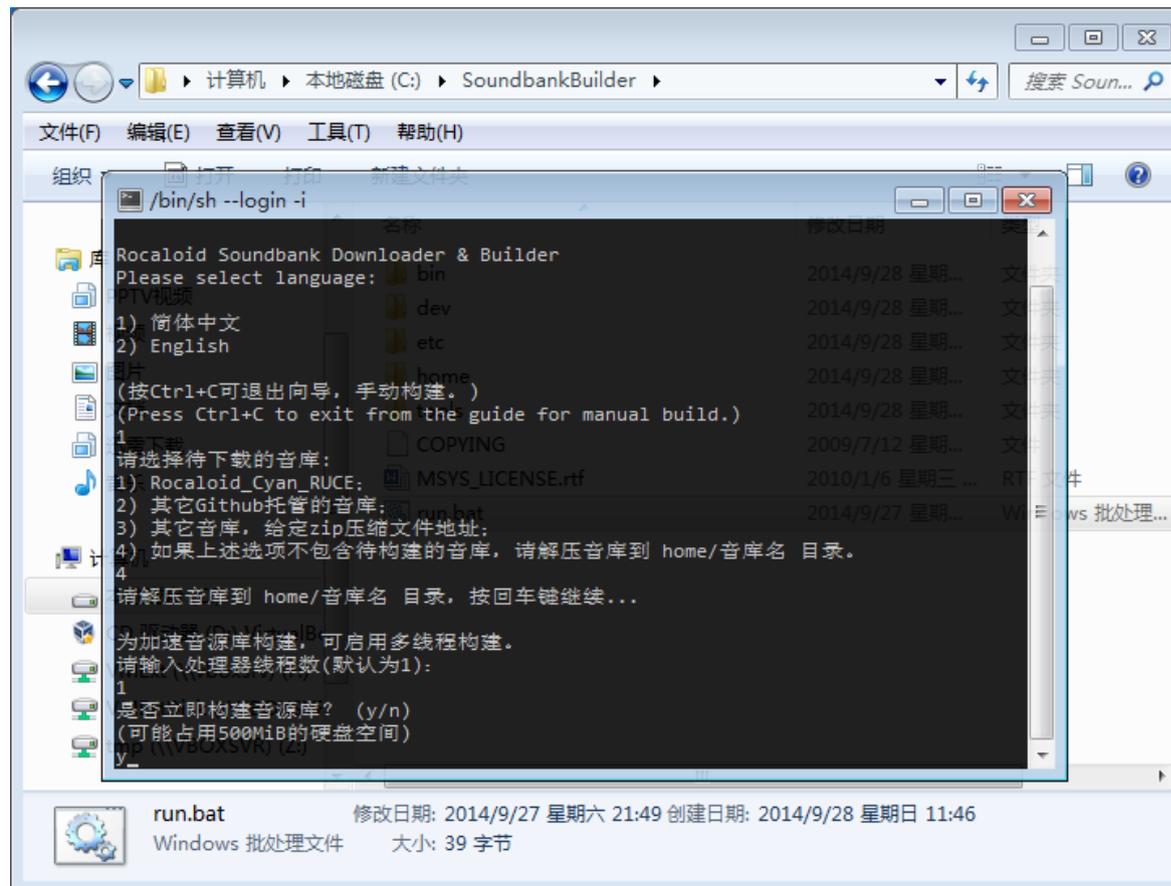
<sup>1</sup> mp3 编码算法受专利保护，故无法被收录于音库开发环境。

<sup>2</sup> SoX 是一个自由的音频编辑软件。

# 构建音库

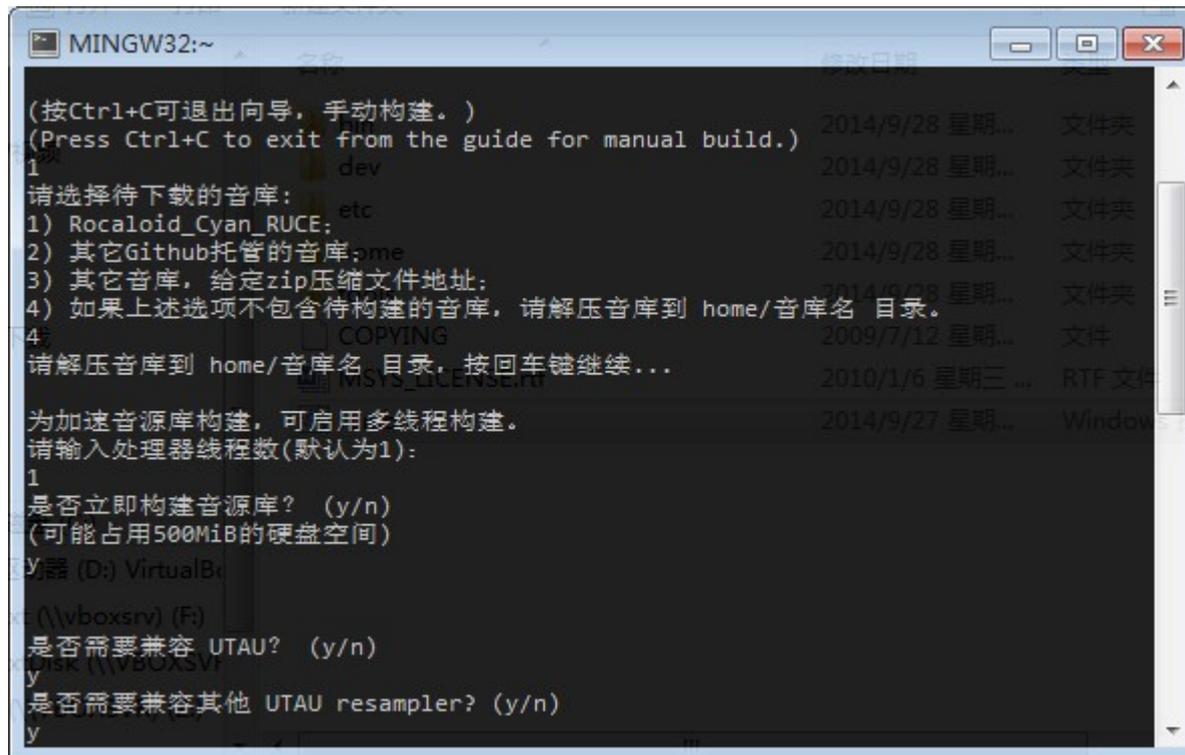
- 为先睹为快，完成上述工作后，我们马上可以调用 Makefile 开始自动化音库构建了。双击批处理文件启动 MSYS 环境：

(会占用约 500MB 磁盘空间，请确认 MSYS 环境所在磁盘空间充足)



# 构建音库

- 由于现阶段 RUCE alpha3 版本只支持 UTAU 或 Cadencii 编辑器，我们需要确认兼容 UTAU，在初次制作音库时勾选上兼容其它 UTAU resampler，这会同时生成 wav 文件。
- 下个主要发布版本：RUCE1.2.0 beta1 会有独立的 API。

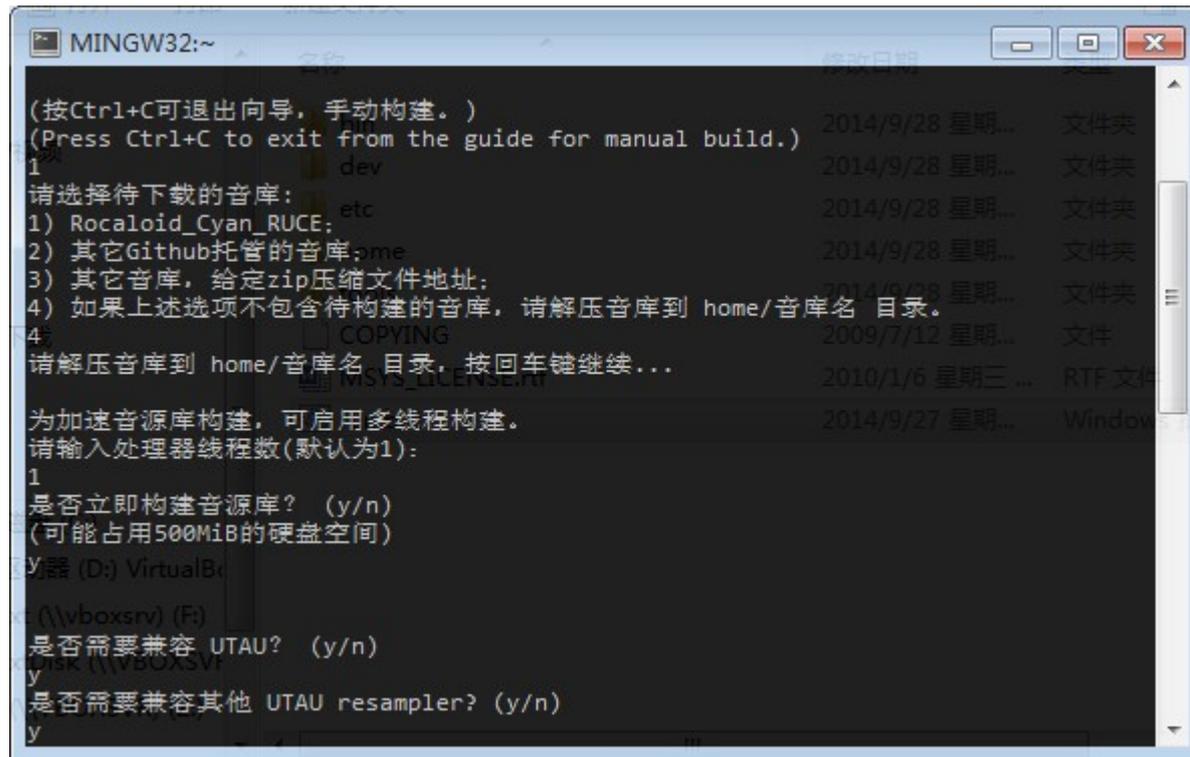


```
MINGW32:~
(按Ctrl+C可退出向导, 手动构建。)
(Press Ctrl+C to exit from the guide for manual build.)
1
dev
请选择待下载的音库:
1) Rocaloid_Cyan_RUCE: etc
2) 其它Github托管的音库: me
3) 其它音库, 给定zip压缩文件地址:
4) 如果上述选项不包含待构建的音库, 请解压音库到 home/音库名 目录。
4
请解压音库到 home/音库名 目录, 按回车键继续...

为加速音源库构建, 可启用多线程构建。
请输入处理器线程数(默认为1):
1
是否立即构建音源库? (y/n)
(可能占用500MiB的硬盘空间)
y
是否需要兼容 UTAU? (y/n)
y
是否需要兼容其他 UTAU resampler? (y/n)
y
```

# 构建音库

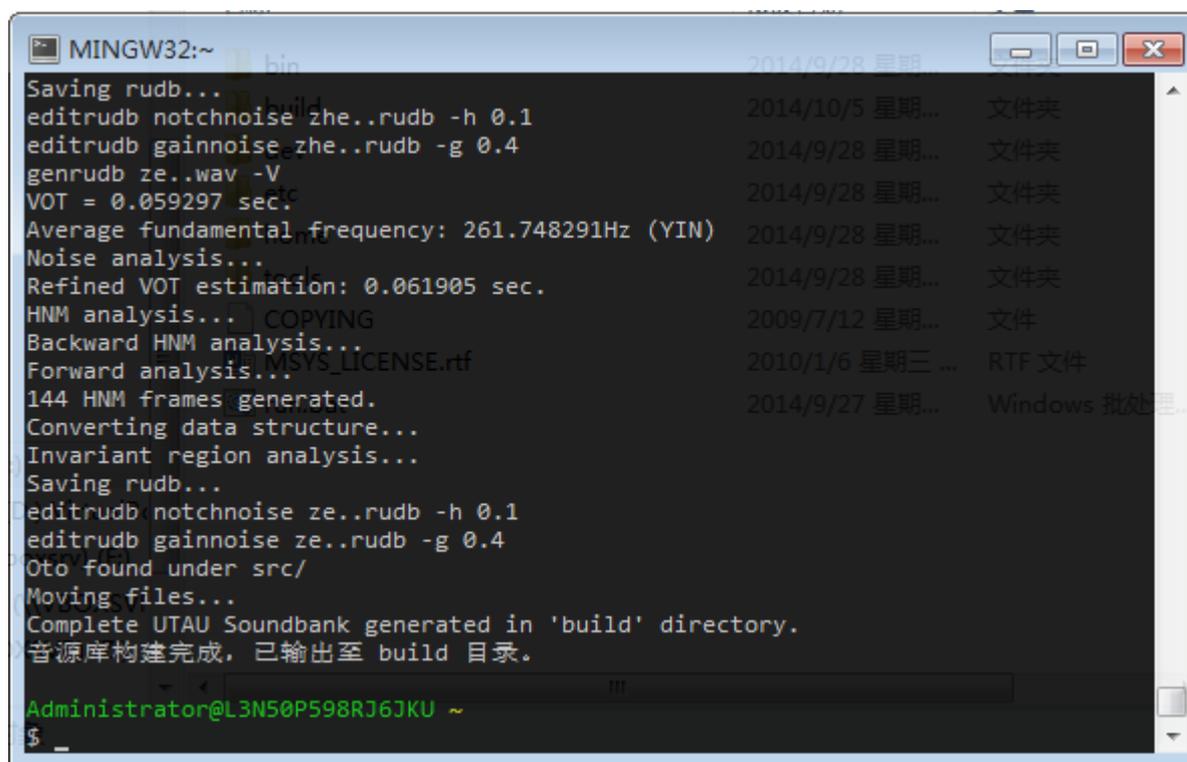
- 构建程序会调用 GNU make ，自动完成音源库的构建。这大约会耗用 5 到 10 分钟的时间。



```
MINGW32:~
(按Ctrl+C可退出向导, 手动构建。)
(Press Ctrl+C to exit from the guide for manual build.)
1 dev
请选择待下载的音库:
1) Rocaloid_Cyan_RUCE;
2) 其它Github托管的音库;
3) 其它音库, 给定zip压缩文件地址;
4) 如果上述选项不包含待构建的音库, 请解压音库到 home/音库名 目录。
4:
请解压音库到 home/音库名 目录, 按回车键继续...
为加速音源库构建, 可启用多线程构建。
请输入处理器线程数(默认为1):
1
是否需要立即构建音源库? (y/n)
(可能占用500MiB的硬盘空间)
y
是否需要兼容 UTAU? (y/n)
y
是否需要兼容其他 UTAU resampler? (y/n)
y
```

# 构建音库

- 构建完成后将如图所示，生成的音库在 MSYS 环境的 build 目录下。



```
MINGW32:~
Saving ru db...
editru db notchnoise zhe..ru db -h 0.1
editru db gainnoise zhe..ru db -g 0.4
genru db ze..wav -V
VOT = 0.059297 sec.
Average fundamental frequency: 261.748291Hz (YIN)
Noise analysis...
Refined VOT estimation: 0.061905 sec.
HNM analysis... COPYING
Backward HNM analysis...
Forward analysis...
144 HNM frames generated.
Converting data structure...
Invariant region analysis...
Saving ru db...
editru db notchnoise ze..ru db -h 0.1
editru db gainnoise ze..ru db -g 0.4
Oto found under src/
Moving files...
Complete UTAU Soundbank generated in 'build' directory.
音源库构建完成, 已输出至 build 目录.

Administrator@L3N50P598RJ6JKU ~
$ _
```

# 之后做什么？

- 在进一步修改调整音库之前，为了能使音库顺利被 UTAU 调用，应当制作 oto.ini 文件。
- 模板 Makefile 自动地生成了一个 oto.ini，但是它不是很精确，需要人工调整，方法同一般 UTAU 音库的制作，请参考相关 UTAU 教程。
- RUCE 不需要 oto.ini 中的固定长度和前后空白参数。
- 将调整好的 oto.ini 复制到音库源码路径的 src 目录下，下次构建音库时 Makefile 会使用调整后的 oto.ini 文件。

# 优化、调整音库

- 目前介绍的只是粗略生成一个 RUCS 音库，很显然我们需要不断调整音库，让它变得更好。
- 接下来几页会先介绍一些背景知识。您会了解到 RUCS 音库的结构、工具链的组成、以及刚才使用到的工具具体做了什么。然后我们将详细讨论如何调整音库。

# RUCE 音库结构

自 alpha3 版本，一个构建完成的 RUCE 音库由下列文件组成：

- 包含每种音节发音的 rudb 文件，对应到录音表里每个发音，存放发音的音频数据；
- PitchModel.json，它是一个 json 格式的文本文件，描述 RUCE 在变调时所做的频谱修正；
- oto.ini( 可选 )，这是 UTAU 使用的音节信息描述文件，计划在 RUCE 1.2.0 beta1 中被移除。

# rudb 文件包含了什么？（可略过）

- rudb 包含的不是波形数据，亦不是对 wav 文件的少量补充，而是可以代替 wav 文件的 HNM 帧数据、清辅音部分的波形、和少量参数。
- HNM(Harmonic Noise Model) 将音频建模为谐波和噪声信号的叠加，其中谐波部分包括各个谐波的瞬间频率、幅度和相位；噪声信号存储为噪声的频谱包络。
- rudb 还包含经过分析，从原 wav 文件中分离出的清辅音部分的时域波形。
- 还有少量的参数用于描述文件、记录时间节点等信息。
- HNM 保证重构出音频和原始音频听觉特性相同，然而数据不同。因此从 rudb 还原出的 wav 文件，听上去和原来声音十分近似，然而它们是不同的。

# Rocaloid 音库工具链

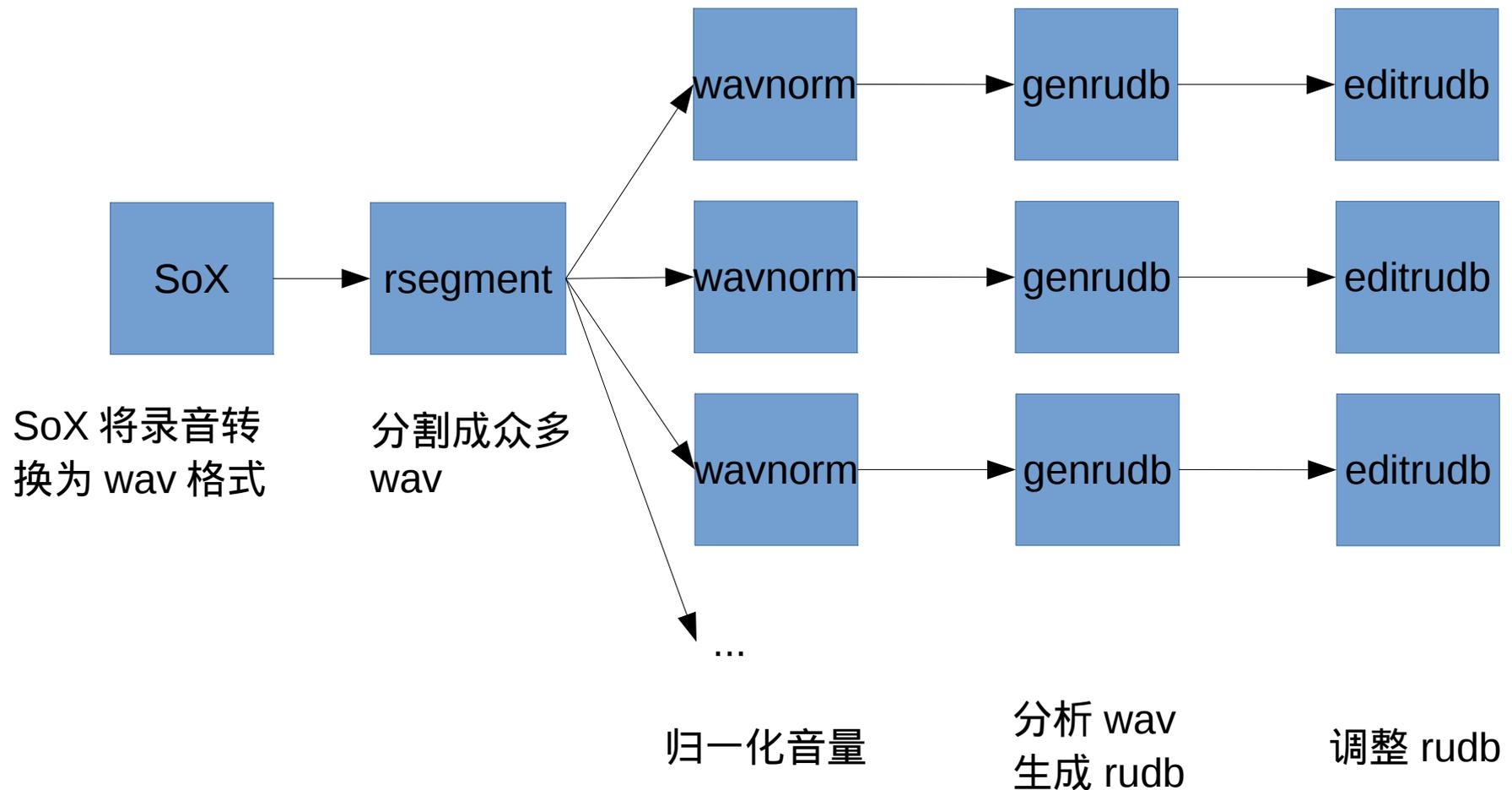
- Rocaloid 音库工具链是一套用于制作和维护 Rocaloid 音源库的命令行工具集。由于在未来会支持 RUCE 以外的其它 Rocaloid 合成引擎，而且它还包括了若干涉及音库其它功能的工具，因此它不叫作“RUCE 音库工具链”。
- 目前包含组件：
  - rsegment – wav 音频切割工具
  - wavnorm – 归一化音频音量
  - genrudb – 从 wav 文件分析生成 ruidb 文件
  - editrudb – 编辑 ruidb 文件
  - proberuidb – 显示 ruidb 文件信息
  - precompdiff – 差分预压缩工具
  - utau2rec – 根据 UTAU 音库生成录音表文件
  - ruidb2oct – 从 ruidb 文件生成 octave 矩阵数据格式
- 较常用的是前五个工具

# 我们刚刚做了什么？

- 批处理文件启动了 MSYS 环境，然后我们看到的命令行运行的其实并不是批处理程序，而是一种 Unix Shell 脚本，它的功能与批处理类似，但更加强大。
- 这个 Shell 写的向导名称叫 get-soundbank，它可以下载音库，解压，自动找到 Makefile 所在的目录，然后调用 make 程序。
- 在 make 自动调用 Rocaloid 音库工具链完成构建后，make 将音库安装在另一个 build 目录下，但是这个 build 目录在音库目录下。
- Shell 将音库的 build 目录的文件移动到 MSYS 目录下的 build 目录。

# 我们刚刚做了什么？

- 根据 Makefile ， make 程序的工作流程是：



# 我们刚刚做了什么？

- 在 rsegment 分割完成后，由于前期和后期处理有多个步骤，所以事实上针对每个发音会生成多种中间文件，它们的后缀如同 .wav, .wav1, .wav2, .rudb, .rudb1, .rudb2...
- 我们提倡使用 wav/rudb 加数字的后缀命名方式。 .wav1 是最初分割产生的文件， .wav2 是第一次处理后的文件，以此类推，最后一次处理将生成 .wav 文件。
- 类似地， .rudb1 是 genrudb 生成的文件， .rudb2 是第一次处理后的 rudb 文件，最后一次处理生成 .rudb 文件。

# 编辑 Makefile

- 我们不推荐编辑音库目录下的 Makefile 文件，而建议编辑 src 目录下的 premakes.make 和 postmakes.make 文件。这是为了让所有 RUCE 音库尽量统一，方便构建。
- Makefile 编写入门（参考）：  
<http://wiki.ubuntu.org.cn/%E8%B7%9F%E6%88%91%E4%B8%80%E8%B5%B7%E5%86%99Makefile>
- Makefile 中行首缩进一定是 Tab，而不是空格。
- 我们推荐你使用一个具有代码着色的编辑器，使文件更清楚可读，例如 Notepad2, Notepad++, Gedit 等.....

# Makefile 快速入门

- postmakes.make 中一个非常典型的例子：

```
gan.rudb : gan.rudb1
```

```
    @ cp fei.rudb1 fei.rudb
```

```
        editrudb param fei.rudb -r 1.048
```

```
        editrudb gainnoise gan.rudb -g 0.4
```

- 这段规则中，gan.rudb 是生成的文件，冒号代表该文件的生成依赖于冒号右边的文件，即 gan.rudb1。下面的三行命令是具体步骤。在行首加上 @ 符号，执行时将不显示命令。
- 若 gan.rudb1 不存在或不是最新，它将去寻找 gan.rudb1 的构建规则并执行；若满足上述条件，它将依次执行下面三行命令：把 fei.rudb1 拷贝到 fei.rudb，然后编辑 fei.rudb。

# Makefile 快速入门

- 有时我们需要对一类文件作处理，例如都是以 b 打头命名的文件：

```
b%.rudb : b%.rudb1
```

```
    @ cp b$*.rudb1 $@
```

```
    editrudb gainnoise $@ -g 0.4
```

- % 是一个通配符，可以代替任何文件名中的任何数量的字符；\$\* 会被 make 替换成 % 所代替的内容；\$@ 会被替换成完整的文件名。
- 例如对于 ban.ru**db**，上述规则等同于：

```
ban.rudb : ban.rudb1
```

```
    @ cp ban.rudb1 ban.rudb
```

```
    editrudb gainnoise ban.rudb -g 0.4
```

# Makefile 快速入门

- 在 premakes.make 和 postmakes.make 中有缺省的设置，完全使用通配符进行匹配：

```
%.wav2 : %.wav1
```

```
    @ cp $*.wav1 $@
```

```
    @ wavnorm $@ -g 1.5 -i -40 -t -s 0.01 -e 0.01
```

```
%.rubd : %.rubd1
```

```
    @ cp $*.rubd1 $@
```

```
    editrubd notchnoise $@ -h 0.1
```

```
    editrubd gainnoise $@ -g 0.4
```

- 因此即使完全套用模板，Makefile 也能使用缺省的设置构建音库。
- 增加“更细节”的规则会自动覆盖掉“更泛化”的规则。
- 上述只是 Makefile 语法的非常小的一部分。

# 如何优化音库？

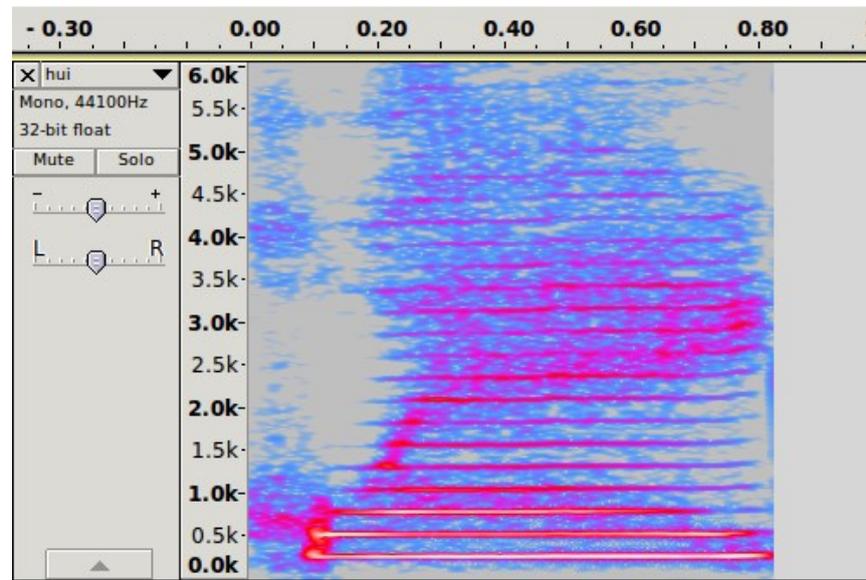
1. 合成一些歌曲，先找到音库的问题——哪个发音存在问题？哪种类型的发音存在问题？
2. 使用音频编辑 / 分析软件，如 Audacity，打开发音文件，找出**具体问题**所在，判断问题原因——例如是切割错误了，还是录音问题，还是工具链调用的某个环节出错等等.....
3. 针对不同问题，修改 premakes.make 和 postmakes.make。前者记录前期处理的规则（操作 wav1/2/3... 文件），后者记录后期处理的规则（操作 rudb1/2/3... 文件）。
4. 删除相关的 wav 和 rudb 文件（但不要删除 wav1 文件），手动调用 make。

# 例子

- 为了让上述流程更明确，我们从一个实际例子出发，逐步演示具体操作过程。
- 关于找出、定位发音问题的方法，以及 Rocaloid 音库工具链的完整使用说明，会在本例之后提及。
- 本例中我们发现，Cyan\_RUCE 音库的“hui”发音不自然。
- 第一步，仔细听合成片段发现问题大致出在该发音的韵尾部分。
- 接下来需要根据听觉现象，进一步找出问题的原因。

# 例子 - 第二步

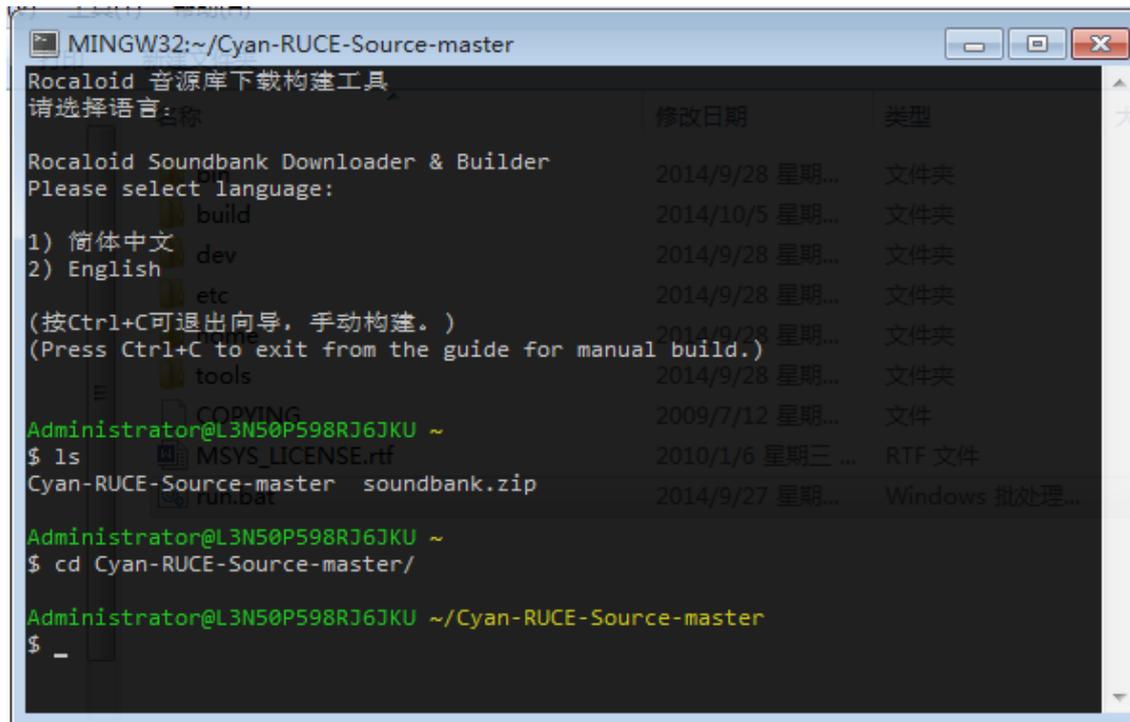
- 第二步，使用 Audacity 打开 hui.wav2，显示频谱图：



- 频谱图很正常，而且听上去也很正常，所以我们可以判定问题不是处在录音或前期处理上。所以问题很可能出在转换，或后期处理的环节。

# 例子 - 第二步

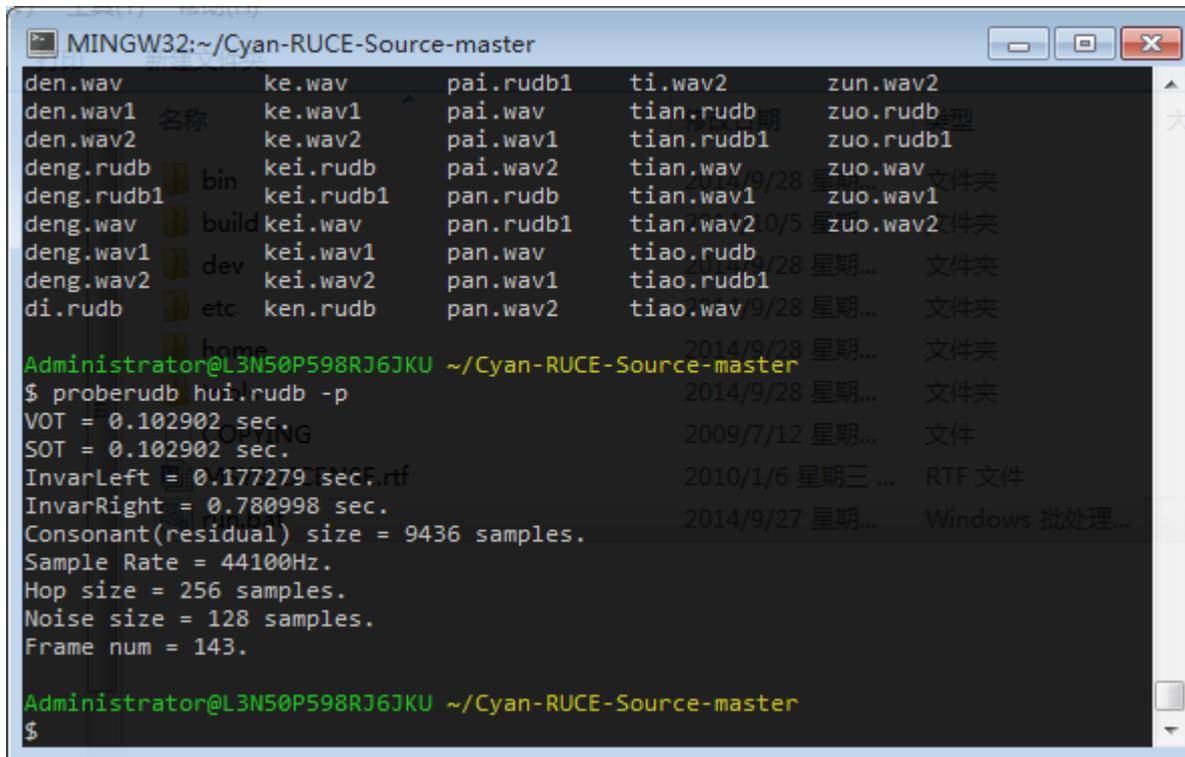
- 我们想检查一下 rudb 文件是否有问题，于是启动 MSYS 音库构建环境，按 Ctrl+C 退出向导，进入 Shell 手动操作。
- 你需要掌握一些 Shell 的基本命令，好在这些学起来很快。
- 使用 ls 查看当前目录，使用 cd 进入音库目录。



```
MINGW32:~/Cyan-RUCE-Source-master
Rocaloid 音源库下载构建工具
请选择语言:
Rocaloid Soundbank Downloader & Builder
Please select language:
  build
1) 简体中文
2) English
etc
(按Ctrl+C可退出向导, 手动构建。)
(Press Ctrl+C to exit from the guide for manual build.)
tools
Administrator@L3N50P598RJ6JKU ~
$ ls
Cyan-RUCE-Source-master  soundbank.zip
Administrator@L3N50P598RJ6JKU ~
$ cd Cyan-RUCE-Source-master/
Administrator@L3N50P598RJ6JKU ~/Cyan-RUCE-Source-master
$ _
```

# 例子 - 第二步

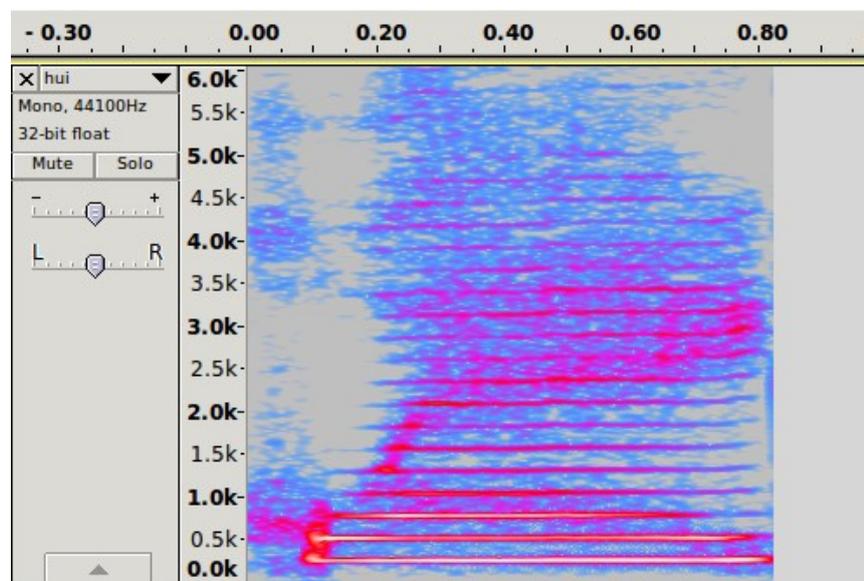
- 你可以再 ls 一次，确认这是存放各种中间文件的目录，然后输入 `proberudb hui.rudb -p`
- 作用是输出 rudb 文件的一些参数。



```
MINGW32:~/Cyan-RUCE-Source-master
den.wav      ke.wav      pai.rudb1   ti.wav2     zun.wav2
den.wav1    名称      ke.wav1     pai.wav     tian.rudb   zuo.rudb   型
den.wav2    ke.wav2     pai.wav1    tian.rudb1  zuo.rudb1
deng.rudb   bin        kei.rudb    pai.wav2    tian.wav    zuo.wav    文件夹
deng.rudb1  kei.rudb1  pan.rudb    tian.wav1   zuo.wav1
deng.wav    build     kei.wav     pan.rudb1   tian.wav2   zuo.wav2   文件夹
deng.wav1   dev       kei.wav1    pan.wav     tiao.rudb   文件夹
deng.wav2   kei.wav2   pan.wav1    tiao.rudb1  文件夹
di.rudb     etc       ken.rudb    pan.wav2    tiao.wav    文件夹
Administrator@L3N50P598R36JKU ~/Cyan-RUCE-Source-master
$ proberudb hui.rudb -p
VOT = 0.102902 sec.
SOT = 0.102902 sec.
InvarLeft = 0.177279 sec.rtf
InvarRight = 0.780998 sec.
Consonant(residual) size = 9436 samples.
Sample Rate = 44100Hz.
Hop size = 256 samples.
Noise size = 128 samples.
Frame num = 143.
Administrator@L3N50P598R36JKU ~/Cyan-RUCE-Source-master
$
```

# 例子 - 第二步

- 我们看到 hui.rudb 中 InvarRight 参数的值是 0.780998 秒。这个参数指定了元音稳定时间的右端点 (Invariant Right 的略写)，即韵尾起始时间。
- 而刚才看到的频谱图中韵尾其实是在大约 0.69 秒的位置：



# 例子 - 第二步

- 0.1 秒的误差对于韵尾时间来说是很大的。因此可以判定是韵尾时间自动检测出错了。
- 韵尾时间的检测由 `genru db` 在转换步骤完成。至此我们判定了问题出现的具体步骤。
- `genru db` 本身只能自动判定韵尾时间，但我们可以后期处理步骤使用 `editru db` 工具修改 `InvariantRight` 参数。

# 例子 - 第三步

- 下面我们要在 `postmakes.make` 中增加一条针对 `hui.rudb` 的后期处理规则。
- `hui.rudb1` 是从 `hui.wav` 转换过来的；而 `hui.rudb` 是从 `hui.rudb1` 后期处理得到。 `postmakes.make` 中有一个默认设置的规则：

```
%.rudb : %.rudb1
```

```
    @ cp $*.rudb1 $@
```

```
        editrudb notchnoise $@ -h 0.1
```

```
        editrudb gainnoise $@ -g 0.4
```

- 这大致是对清辅音部分做了些修改，与 `InvariantRight` 无关。

# 例子 - 第三步

- 我们复制一份默认规则，并把通配符改成 hui：

```
hui.rudb : hui.rudb1
```

```
@ cp hui.rudb1 hui.rudb
```

```
editrudb notchnoise hui.rudb -h 0.1
```

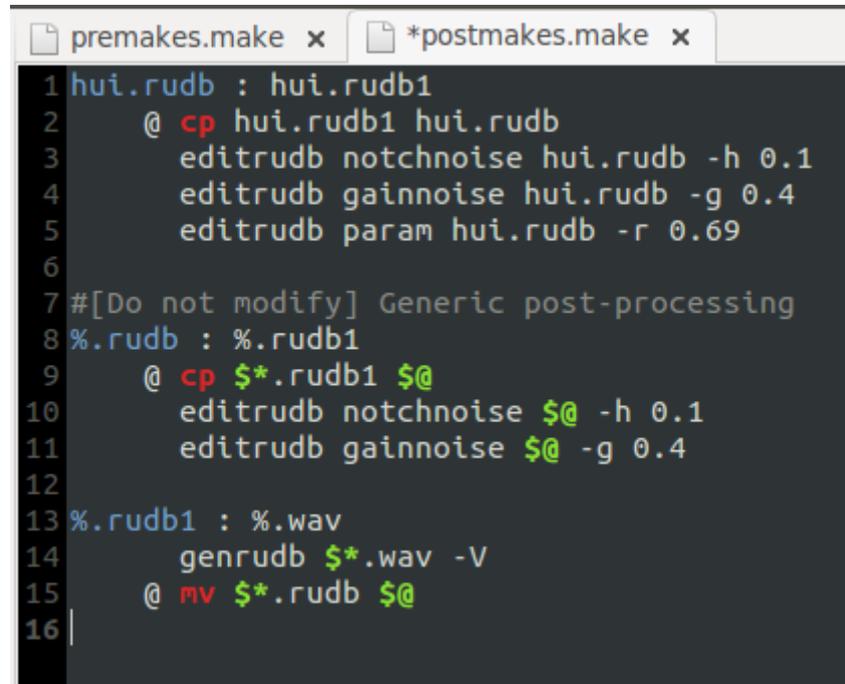
```
editrudb gainnoise hui.rudb -g 0.4
```

- 为了修改 InvariantRight 参数，我们在最后加入一行（行首缩进请以一个 Tab 开始）：

```
editrudb param hui.rudb -r 0.69
```

# 例子 - 第三步

- 最终修改的 postmakes.make 如图所示：

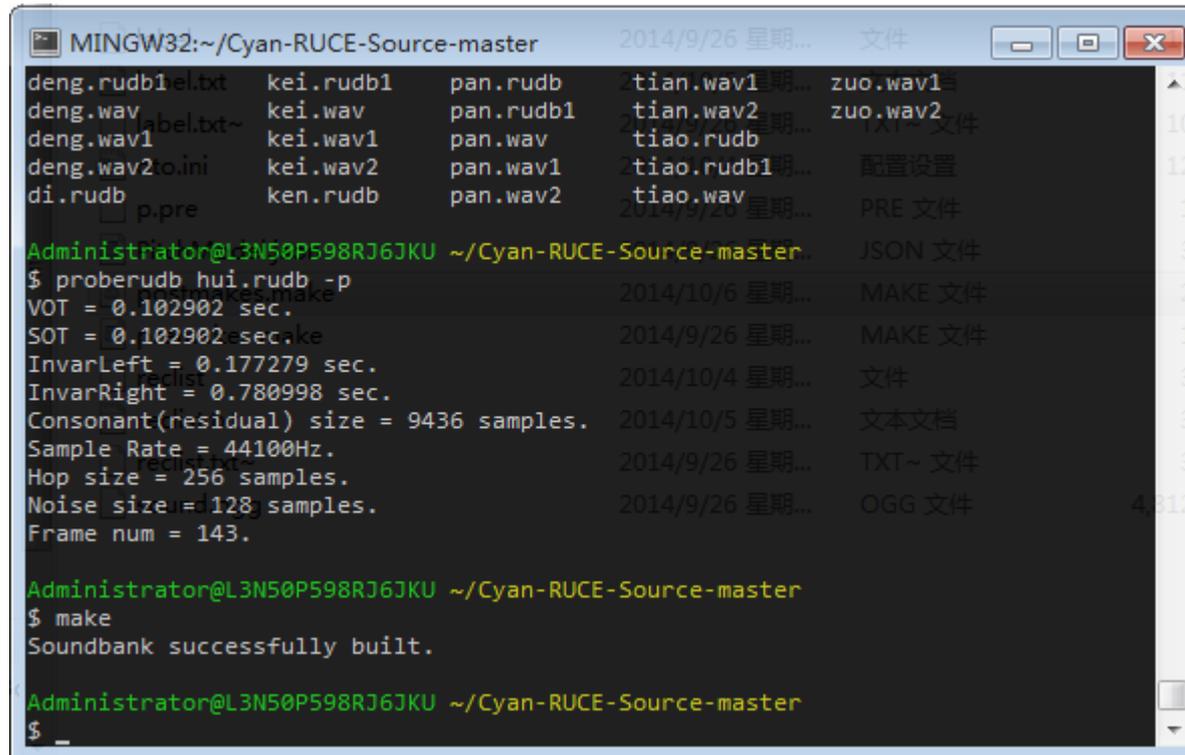


```
premakes.make x *postmakes.make x
1 hui.rudb : hui.rudb1
2   @ cp hui.rudb1 hui.rudb
3   editrudd notchnoise hui.rudb -h 0.1
4   editrudd gainnoise hui.rudb -g 0.4
5   editrudd param hui.rudb -r 0.69
6
7 #[Do not modify] Generic post-processing
8 %.rudb : %.rudb1
9   @ cp $*.rudb1 $@
10  editrudd notchnoise $@ -h 0.1
11  editrudd gainnoise $@ -g 0.4
12
13 %.rudb1 : %.wav
14   genrudd $*.wav -V
15   @ mv $*.rudb $@
16 |
```

- 保存 postmakes.make 。

# 例子 - 第四步

- 如果直接 make ，不会发生任何事。因为 make 程序检测到 hui.rudb 比 hui.rudb1 的修改时间更新，所以不会重新进行后期处理。



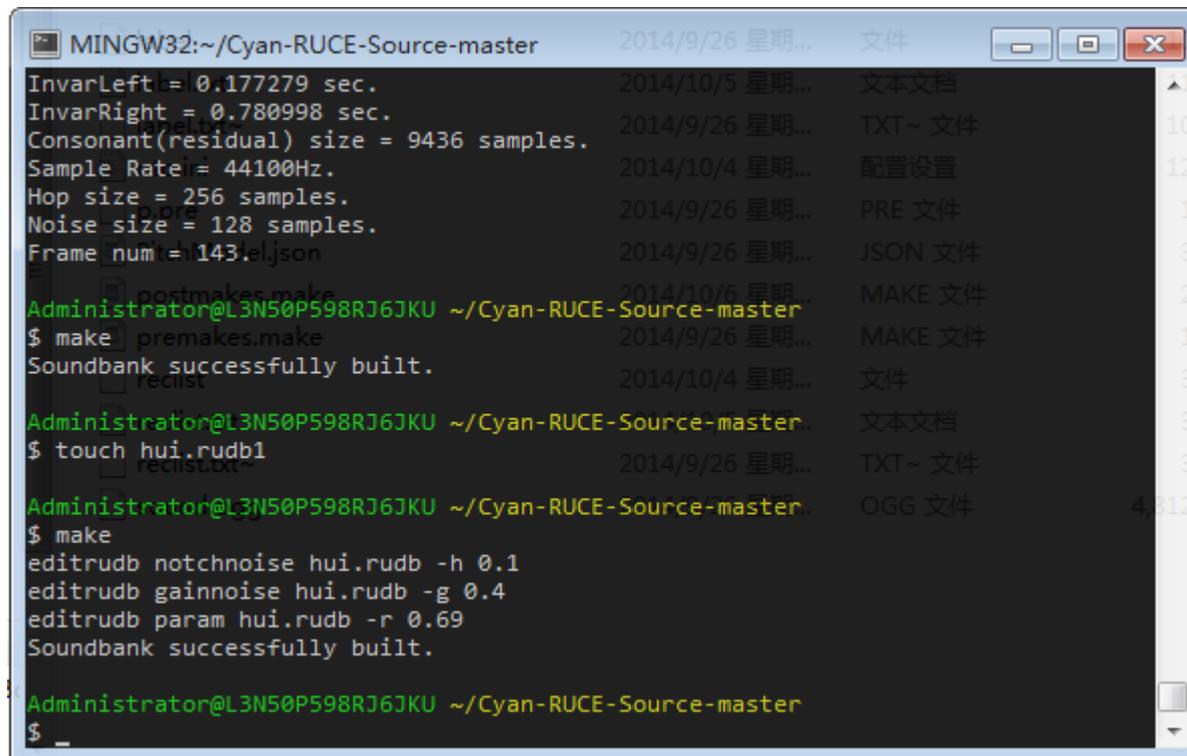
```
Administrator@L3N50P598RJ6JKU ~/Cyan-RUCE-Source-master
$ proberudb hui.rudb -p
VOT = 0.102902 sec.
SOT = 0.102902 sec.
InvarLeft = 0.177279 sec.
InvarRight = 0.780998 sec.
Consonant(residual) size = 9436 samples.
Sample Rate = 44100Hz.
Hop size = 256 samples.
Noise size = 128 samples.
Frame num = 143.

Administrator@L3N50P598RJ6JKU ~/Cyan-RUCE-Source-master
$ make
Soundbank successfully built.

Administrator@L3N50P598RJ6JKU ~/Cyan-RUCE-Source-master
$
```

# 例子 - 第四步

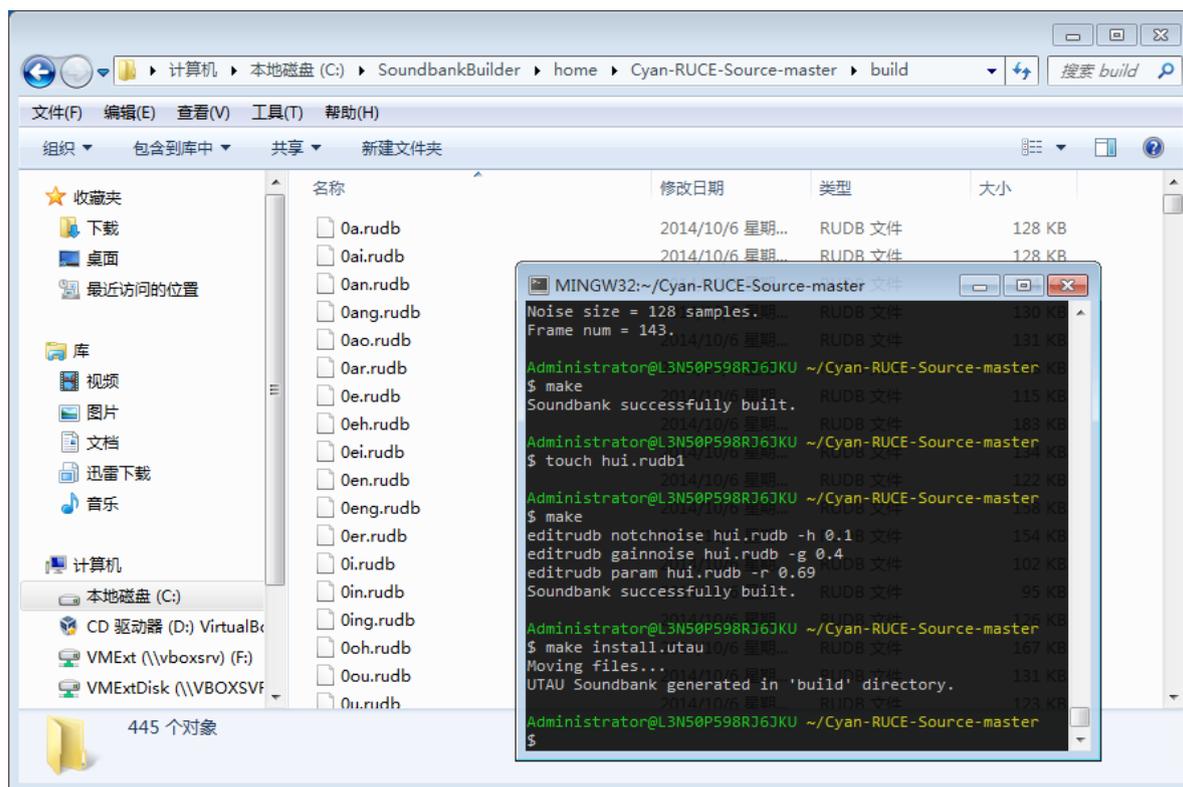
- 因此我们要手动触发这个处理规则。
- 可以使用 `rm hui.rudb` 删除 `rudb` 文件，或者 `touch hui.rudb1` 使 `rudb1` 文件比 `rudb` 文件更新。然后再 `make`。



```
MINGW32:~/Cyan-RUCE-Source-master 2014/9/26 星期... 文件
InvarLeft = 0.177279 sec.
InvarRight = 0.780998 sec.
Consonant(residual) size = 9436 samples.
Sample Rate = 44100Hz.
Hop size = 256 samples.
Noise size = 128 samples.
Frame num = 1431.
Administrator@L3N50P598RJ6JKU ~/Cyan-RUCE-Source-master
$ make premakes.make
Soundbank successfully built.
Administrator@L3N50P598RJ6JKU ~/Cyan-RUCE-Source-master
$ touch hui.rudb1
Administrator@L3N50P598RJ6JKU ~/Cyan-RUCE-Source-master
$ make
editrudb notchnoise hui.rudb -h 0.1
editrudb gainnoise hui.rudb -g 0.4
editrudb param hui.rudb -r 0.69
Soundbank successfully built.
Administrator@L3N50P598RJ6JKU ~/Cyan-RUCE-Source-master
$
```

# 例子 - 第四步

- 至此我们已经修好了 hui 这个发音。
- 但是输出的文件是在音库目录。
- 输入 `make install.utau` , 它把 UTAU 音库所需要的文件都拷贝到音库目录下的 `build` 目录。至此我们的例子就完成了。



# 多个发音的矫正

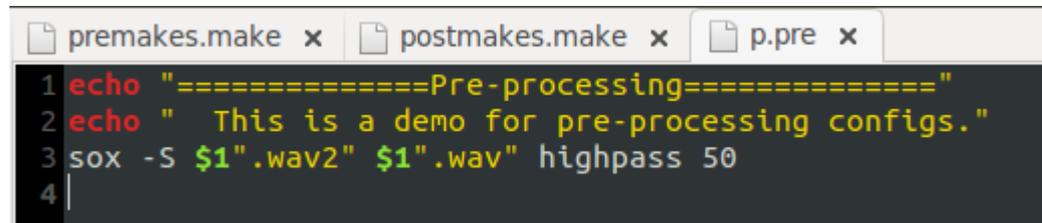
- 有时候出现问题的是一组发音。
- 例如辅音 p 打头的发音，在开头都或多或少有一个爆音，表现为低音特别重。
- 尽管可以使用一个类似 p%.wav 的规则解决问题，但是每次修改这条规则都要手动清除 p\*.wav 文件，可能较为繁琐。
- 解决方法是，把对 p\*.wav 的操作写进一个单独的 Shell 脚本文件里，并让 p\*.wav 的规则依赖于这个脚本文件。
- 脚本文件被改动后，所有依赖于它的 wav 文件也会被 make 重制。

# 多个发音的矫正

- 我们在 Cyan\_RUCE 的 premakes.make 中可以可看到：

```
9 #Final
10 p%.wav : p%.wav2 src/p.pre
11     @ sh src/p.pre p$*
12
```

- 在 src/p.pre 中有：



```
premakes.make x postmakes.make x p.pre x
1 echo "====Pre-processing===="
2 echo " This is a demo for pre-processing configs."
3 sox -S $1".wav2" $1".wav" highpass 50
4 |
```

- 这代表对 p\*.wav2 进行 50Hz 的高通滤波，输出到 p\*.wav。

# Rocaloid 音库工具链

- 下面几页介绍 Rocaloid 音库工具链的各个组件使用方法。
- 将包括 rsegment, wavnorm, genruadb, editruadb, proberuadb, 和 utau2rec 。
- 剩余几个组建由于不常用，不被包含在本教程中。欲了解请查阅源代码。

# rsegment

- wav 录音分割工具
- 调用格式： rsegment <wav 文件 > < 标签轨文件 > < 录音表文件 > [-r]  
( <> 括起内容表示必填参数， [] 括起内容为可选参数 )
- 将 wav 录音文件按标签轨和录音表分割成若干个以录音表内音节名命名的 wav 文件。
- 加 -r 参数时，不生成 wav 文件，而生成名为 RegeneratedTrack.txt 的标签轨，可导入 Audacity 用于对照标注是否正确。

# wavnorm

- wav 文件归一化音量、静音切除工具
- 调用格式： wavnorm <wav 文件> [-t] [-s 噪音起始门限] [-e 末端噪音门限] [-g 增益] [-i 最大强度] [-z 分析窗大小] [-v] [-V]
- -t：加此参数切除静音部分
- -s 噪音起始门限：设定 wav 文件头部噪音部分的判定幅度，默认 0.005。
- -e 末端噪音门限：类似 -s，设定尾部噪音判定幅度，默认 0.01。

# wavnorm

- -g 增益：在限定最大音量前增强的倍数，默认 1.5。
- -i 最大强度：所限定的最大音量，单位分贝，默认 -50。（因为输入在 0~1 之间，所以分贝数是负的）
- -z 分析窗大小：分析音量时每帧的时间跨度，单位采样，默认 1024。
- -v：输出 wavnorm 版本。
- -V：在运行中输出调试信息。

# genrub

- 分析转换 wav 文件到 rub 音库数据文件
- 调用格式：`genrub <wav 文件> [-u 基频上限] [-l 基频下限] [-m 基频提取算法] [-s 谐波成分频率上限] [-h 分析跳跃长度] [-z 分析窗大小] [-w 分析窗类型] [-c 不规则谐波剔除阈值] [-t 发音起始时间] [-i 元音稳定区域判断阈值] [-v] [-V]`
- -u 基频上限：基频判定的最大值，单位赫兹，默认 700。
- -l 基频下限：基频判定的最小值，单位赫兹，默认 80。
- -m 基频提取算法：目前支持 YIN 和 SPECSTEP，默认 YIN。

# genrudb

- -s 谐波成分频率上限：ru db 中谐波成分的频率上限，单位赫兹，默认 10000。
- -h 分析跳跃长度：每隔此距离进行一次分析，单位采样，默认 256。
- -z 分析窗大小：分析窗时长，单位采样，默认 2048。
- -w 分析窗类型：支持 hanning（汗宁窗）、hamming（海明窗）、和 blackman（布莱克曼窗），默认 hanning。
- -c 不规则谐波剔除阈值：当某个谐波距离基频整数倍偏离超过此频率时，这个谐波就会被剔除，单位赫兹，默认值 30。

# genrddb

- -t 发音起始时间：手动制定发音起始时间（VOT），而跳过自动分析，单位：秒，默认是自动分析。
- -i 元音稳定区域判断阈值：元音稳定区域的判定阈值，越大则标准越严格，元音稳定时间越短，默认 0.003。
- -v：输出 genrddb 版本。
- -V：在运行中输出调试信息。

# editrudb

- rldb 编辑工具
- 调用格式： editrudb < 命令 > < 命令相关格式 >
- editrudb 是一个特殊的工具，它有多种不同的功能，由第二个参数指定，而第三个参数针对不同命令有不同格式。
- editrudb 仍在开发中，目前只有一些最基本的功能支持。
- 命令包括： param, gainnoise, notchnoise 。

# editruadb param

- 设定 ruadb 文件的参数
- 调用格式： `editruadb param <ruadb 文件> [-t 节奏对其时间] [-s 周期性起始时间] [-r 韵尾起始时间] [-l 韵头结束时间]`
- -t 节奏对其时间：控制音节与音符起始的对其点时间，单位为秒，默认值是 ruadb 的发音起始时间。
- -s 周期性起始时间： ruadb 所基于的 wav 文件的周期性起始时间，即发音起始时间 (VOT)<sup>1</sup>。
- -r 韵尾起始时间： ruadb 所基于的 wav 文件的音节韵尾起始时间，单位为秒，即 InvarRight。
- -l 韵头结束时间：类似韵尾起始时间，即 InvarLeft。

<sup>1</sup>VOT(Voice Onset Time) 具有多种定义，具体意思应当参照应用场合。

# editrudb gainnoise

- 增强或减弱清辅音成分
- 调用格式： `editrudb gainnoise <rudb 文件> [-g 增益倍数]`
- -g 增益倍数： `rudb` 文件中的清辅音成分将被增强此倍数，当此数值在 0 到 1 之间时，清辅音成分被削弱。

# editrudb notchnoise

- 以指定中心减弱 ru db 中清辅音成分
- genru db 分析过程中有时清辅音会提取不干净，在 VOT 处可能有少量周期成分残留，使用 notchnoise 减弱这些残留。
- 调用格式：`editru db notchnoise <ru db 文件> [-s 周期性起始时间] [-r 半径] [-h 中心幅度]`
- -s 周期性起始时间：减弱的中心位置，单位为秒，一般在 VOT 处。
- -r 半径：清辅音减弱的半径，单位为秒，默认为 0.02。
- -h 中心幅度：清辅音减弱的中心幅度倍数，默认为 0.2。

# proberudb

- 以文本方式输出 rudb 文件中的数据
- 调用格式： `proberudb <rudb 文件> [-u] [-p]`
- 当不加任何可选参数时， `proberudb` 将以树状形式输出 rudb 文件中全部的 HNM 帧数据。建议将输出重定向到文件，再用文本编辑器打开：`proberudb x.rudb > dump.txt`，或者用管线使用查看工具打开：`proberudb x.rudb | less`
- `-u`：以 UTAU 的 `oto.ini` 文件格式输出一行参数。
- `-p`：输出 rudb 文件中的参数信息。

# proberudb

proberudb -p 输出格式的解读：

- VOT：发音开始时间，这里是指节奏对其时间。
- SOT：周期性开始时间。
- InvarLeft：左端元音稳定时间，即韵头结束时间。
- InvarRight：右端元音稳定时间，即韵尾结束时间。
- Consonant(residual) size：清辅音残余数据长度。
- Sample Rate：清辅音数据采样频率。
- Hop size：帧跳跃长度。
- Noise size：随机信号包络长度。
- FrameNum：HNM 帧数量。

# 更新工具链

- 随着新的 RUCE 及 Rocaloid 音库工具链的发布，为了能让音库得到更新，需要更新相关组件。
- 获得 RUCE 及 Rocaloid 音库工具链的可执行档 (.exe) 及动态链接库 (.dll) 后，将其解压至 MSYS 环境的 Tools 目录下。
- 在音库目录执行 `make clean` 删除所有生成的文件，再执行 `make` 重新构建音库。

# 进阶教程 – PitchModel.json

- 由于每个人的音色不同，RUCE 进行变调时需要不同的修正参数，从而尽量减少音色的失真。
- 此类参数以纯文本形式记录在 src/PitchModel.json 文件中。
- 这些参数种类繁多，可对音库中任意一个文件，或任意一组文件进行参数指定。
- 由于 RUCE 是一个基于频谱音频处理技术的合成引擎，这些参数主要指定了在不同音高下语音的频谱特征变化。

# PitchModel.json 结构

- PitchModel.json 使用 JSON 格式编写，JSON 是一种类似 XML 的数据交换格式，参考：

<http://www.w3school.com.cn/json/>

<http://www.json.org/>

- Entries 列表包含了任意数量的参数指定规则。
- 每个规则中的 Wildcard “名称 / 值对” 使用 Shell 通配符匹配到任意一个或一类音库中的音节单元；Inherit “名称 / 值对” 代表该规则中未指定的信息均继承自 Wildcard 为该 Shell 通配符的规则。
- 对于每个音节单元名称，最后一个 Wildcard 匹配样式符合的规则将被采用。

# PitchModel.json 规则

- LDecay 与 HDecay 系列参数指定了两个随基频变化的三角形的增益滤波器，用于随音高增加或降低改变频谱能量分布。
- 当基频低于 LDecay\_F0 或高于 HDecay\_F0 时，滤波器被启用。
- LDecay\_Sensitivity 与 HDecay\_Sensitivity 指定了基频每偏离 LDecay\_F0 或 HDecay\_F0 一赫兹，滤波器中心增益的幅度变化。此幅度变化量以自然对数幅值为单位。
- LDecay\_Center 与 HDecay\_Center 指定了该滤波器的中心频率。
- LDecay\_Bound 与 HDecay\_Bound 指定了该滤波器的最高频率。

# 幅度曲线参数

- HmncCurve 与 NoizCurve 数组指定了两组幅度调整量随基频变化的折线变化关系。
- HmncCurve 指定了谐波成分的幅度调整量； NoizCurve 指定了非谐波（气音）成分的幅度调整量。
- 数组格式为 [ 频率， 幅度， 频率， 幅度 ... ]；频率单位为赫兹；幅度调整量单位为自然对数幅度。

# 相位同步曲线

- PhseCurve 格式类似 Hmnc/NoizCurve ，指定了一组相位同步随基频变化的折线变化关系。
- 参考 RUCE 的相位同步参数。