在数据准备部分只需要自己创建三个文件，分别是：utt2spk、text、wav.scp文件，

需要手动创建的文件

文件“text”包含每段发音的标注。

```
s5# head -3 data/train/text
sw02001-A_000098-001156 HI UM YEAH I'D LIKE TO TALK ABOUT HOW YOU DRESS FOR WORK AND
sw02001-A_001980-002131 UM-HUM
sw02001-A_002736-002893 AND IS
```

每行的第一项是发音编号（utterance-id），可以是任意的文本字符串，但是如果在你的设置中还包含说话人信息， 你应该把说话人编号（speaker-id）作为发音编号的前缀。这对于音频文件的排序非常重要。发音编号后面跟着的 是每段发音的标注。你不用保证这里出现的每一个字都出现在你的词汇表中。词汇表之外的词会被映射到 `data/lang/oov.txt` 中。 注意：尽管在这个特别的例子中，我们用下划线分割了发音编号中的“说话人”和“发音”部分，但是通常用破折号（“-”）会更安全 一点。这是因为破折号的ASCII值更小。有人向我指出说，如果使用

下划线，并且说话人编号的长度不一，在某些特殊的情况下， 如果使用标准"C"语言风格对字符串进行排序，说话人编号和对应的发音编号会被排成不同的顺序。 另外一个很重要的文件是**wav.scp**。在Switchboard例子中，

```
s5# head -3 data/train/wav.scp
sw02001-A /home/dpovey/kaldi-trunk/tools/sph2pipe_v2.5/sph2pipe -f wav -p -c 1 /export/co
sw02001-B /home/dpovey/kaldi-trunk/tools/sph2pipe_v2.5/sph2pipe -f wav -p -c 2 /export/co
```

这个文件的格式是

```
<recording-id> <extended-filename>
```

其中，“extended-filename”可能是一个实际的文件名，或者就像本例中所述那样，是一段提取wav格式文件的命令。 extended-filename末尾的管道符号表明，整个命令应该被解释为一个管道。等会我们会解释什么是“recording-id”， 但是首先，我们需要指出，如果“segments”文件不存在，“wav.scp”每一行的第一项就是发音编号。 在Switchboard设置中，我们

有“segments”文件，所以下面我们就讨论一下这个文件。

```
s5# head -3 data/train/segments
sw02001-A_000098-001156 sw02001-A 0.98 11.56
sw02001-A_001980-002131 sw02001-A 19.8 21.31
sw02001-A_002736-002893 sw02001-A 27.36 28.93
```

"segments" 文件的格式是:

```
<utterance-id> <recording-id> <segment-begin> <segment-end>
```

其中，segment-begin和segment-end以秒为单位。它们指明了一段发音在一段录音中的时间偏移量。“recording-id” 和在“wav.scp”中使用的是同一个标识字符串。再次声明一下，这只是一个任意的标识字符串，你可以随便指定。 文件"reco2file_and_channel"只是在你用NIST的sclite工具对结果进行评分（计算错误率）的时候使用：

```
s5# head -3 data/train/reco2file_and_channel
sw02001-A sw02001 A
sw02001-B sw02001 B
sw02005-A sw02005 A
```

格式为：

```
<recording-id> <filename> <recording-side (A or B)>
```

filename通常是.sph文件的名字，当然需要去掉sph这个后缀；但是也可以是任何其他你在“stm”文件中使用的标识字符串。“录音方”（recording side）则是一个电话对话中两方通话（A或者B）的概念。如果不是两方通话，那么为保险起见最好 使用“A”。如果你并没有“stm”文件，或者你根本不知道这些都是什么东西，那么你可能就不需要reco2file_and_channel"文件。 最后一个需要你手动创建的文件是“utt2spk”。该文件指明某一段发音是哪一个说话人发出的。

```
s5# head -3 data/train/utt2spk
sw02001-A_000098-001156 2001-A
sw02001-A_001980-002131 2001-A
sw02001-A_002736-002893 2001-A
```

文件格式是：

```
<utterance-id> <speaker-id>
```

注意一点，说话人编号并不需要与说话人实际的名字完全一致——只需要大概能够猜出来就行。 在这种情况下，我们假定每一个说话方（电话对话的每一方）对应一个说话人。这可能不完全正确—— 有时一个说话人会把电话交给另外一个说话人，或者同一个说话人会在不同的对话中出现——但是上述假定 对我们来说也足够用了。如果你完全没有关于说话人的信息，你可以把发音编号当做说话人编号。那么 对应的文件格式就变为```

```。 在一些实例脚本中还出现了另外一个文件，它在Kaldi的识别系统的建立过程中只是偶尔使用。在Resource Management （RM）设置中该文件是这样的：

```
s5# head -3 ../../rm/s5/data/train/spk2gender
adg0 f
ahh0 m
ajp0 m
```

这个文件根据说话人的性别，将每个说话人编号映射为“m”或者“f”。 上述所有文件都应该被排序。如果没有排序，你在运行脚本的时候就会出现错误。在 \ref io_sec_tables 中我们解释了为什么需要这样。这与（Kaldi的）I/O框架有关，归根到底是因为排序后的文件可以在一些不支持 fseek()的流中——例如，含有管道的命令——提供类似于随机存取查找的功能。需要Kaldi程序都会从其他Kaldi命令 中读取多个管道流，读入各种不同类型的对象，然后对不同输入做一些类似于“合并然后排序”的处理。既然要合并排序， 当然需要输入是经过排序的。小心确保你的shell环境变量LC_ALL定义为“C”。例如，在bash中，你需要这样做：

```
export LC_ALL=C
```

如果你不这样做，这些文件的排序方式会与C++排序字符串的方式不一样，Kaldi就会崩溃。这一点我已经再三强调过了！

kaldi中的yesno主要分为以下几个步骤：

1、data preparation 数据准备

local/prepare_data.sh 准备数据：语音数据，后面跟着语音数据的文件地址

例如：local/prepare_data.sh waves_yesno

waves_yesno 代表语音文件的目录

local/prepare_dict.sh 准备数据字典 lexicon文件 ： 字典文件，存放所有的音素，格式为

<单词> <音素列表>

utils/prepare_lang.sh

local/prepare_lm.sh

language model 直接local/prpare_lm.sh运行即可

2、feature extraction 特征提取

steps/make_mfcc.sh
```

steps/make_mfcc.sh --nj 1 目标文件路径 目标文件mfcc存放路径 用户指定文件夹(mfcc)

steps/compute_cmvn_stats.sh 用来计算mfcc的m均值 v方差 n归一化

steps/compute_cmvn_stats.sh 目标文件路径 目标文件mfcc的cmvn存放路径 用户指定文件夹(mfcc)

utils/fix_data_dir.sh 用来校验数据准备的是否完备

3、mono trainging 训练阶段

steps/train_mono.sh --nj 1 --cmd "命令" data路径 datalang路径 expmono路径

4、graph compliation

utils/mkgraph.sh

5、decoding 预测阶段

steps/decode.sh graphdir datadir srcdir