

Tcl, Git and Makefile

EE291F

Speaker Yuxuan Li (liyx22025@shanghaitech.edu.cn)

VSPLab@SIST



上海科技大学

ShanghaiTech University

目录

1 Tcl

2 Git

3 Makefile

4 项目组织

Tcl

我们如何使用 EDA 软件

数字后端的 EDA 工具往往会提供以下三种交互方式

- 图形界面（GUI）：完备但很少使用，通常仅在查看波形和版图时打开用于观察。
- Tcl 命令：你可以一行一行地输入 Tcl 命令来进行交互式操作（类似 `python`）。
- Tcl 脚本：你可以写好一个 Tcl 脚本让工具运行（类似 `python main.py`）。

或许和大家想象的不同，**数字后端的 EDA 软件更多的是通过脚本和命令的方式来使用**。

立志成才 报国裕民

Tcl 是什么?

Tcl 的全称是**工具命令语言**（Tool Command Language），是一门可嵌入的通用脚本语言。

Tcl 在 EDA 工具中得到了广泛的支持，这有多方面的原因

- 语法简单：电路设计者不是专业的程序员，Tcl 相较于其他脚本语言学习成本低。
- 扩展容易：Tcl 的设计哲学是一切皆命令，EDA 软件很容易定义自己的工具命令。
- 嵌入便捷：Tcl 是一个很小的 C 库[1]，EDA 软件只要链接它就能得到一套交互命令环境。

[1]库的名称是 `libtcl`

立志成才 报国裕民

Tcl 是什么?

Tcl 有一个通用的解释器 tclsh，它的用法和 python 很相似，都可以交互运行或脚本运行。

通过 tclsh 交互运行 Tcl 命令

```
tclsh
```

通过 tclsh 运行 Tcl 脚本

```
tclsh main.tcl
```

你之后可以将 tclsh 换成 genus、innovus、dc_shell、icc2_shell、fm_shell 等，这样就可以通过 Tcl 操作 EDA 软件了（命令行选项可能略有不同）。它们本质上都是完全相同的 Tcl 解释器，但每个工具会定义不同的 Tcl 命令来包装该工具内的各个操作（Tcl 只是一个包装!）。

立志成才 报国裕民

建立一个项目目录

在一台服务器上开始工作，我们需要建立一个工作区，并为每个项目创建一个目录。在先前的课程 About Linux 中，我们已经学习过如何创建 /home/workspace 作为工作区[2]，那么

```
cd /home/workspace
mkdir Demo
```

随后进入你刚刚创建的 Demo 项目目录

```
cd Demo
```

当敲命令时，请充分思考自己在做什么并观察报错输出，而不是机械化的向终端复制粘贴你看到的每一条命令。例如，假如这里 /home/workspace 还未创建，上述操作显然是会失败的。

[2]你也可以使用 ~/workspace 之类的路径作为工作区，这都是可以的。

立志成才 报国裕民

Tcl 的输出

你可以创建一个 main.tcl，并尝试写入

```
1 puts "hello world"
```

Tcl 使用 puts 作为输出命令，字符串在不包含空格的情况下可以省略双引号。

立志成才 报国裕民

Tcl 的变量

Tcl 的变量通过 set 定义，变量定义时直接写 VARIABLE 但使用时要写 ${VARIABLE}

```
1 set VARIABLE hi
2 puts ${VARIABLE}
```

- 推荐用大写和下划线的变量命名 STUDENT_ID，尽管 student_id 和 studentId 也合法。
- 推荐使用变量时总是用 ${VARIABLE}，尽管 $VARIABLE 多数情况下也是合法的。

Tcl 的列表

Tcl 的列表用 `[list ...]` 表示，可以用 `foreach` 遍历

```
set LIST [list alpha beta gamma]
foreach ELEMENT ${LIST} {
    puts ${ELEMENT}
}
```

- 通常采用四个空格作为缩进，尽管任何缩进都是合法的。
- 通常左大括号不换行，尽管换行也是合法的。

立志成才 报国裕民

Tcl 的命令

Tcl 的命令用 proc 定义，作为参数的变量一般用小写和下划线命名

```
proc vsp_say {str} {
    puts "vsp: ${str}"
}
```

```
vsp_say "welcome to ee291f"
```

你也可以将上述 proc 放到一个独立的文件 vsp.tcl，并在 main.tcl 中包含它

```
source vsp.tcl
```


立志成才 报国裕民

Tcl 的 Shell 调用

Tcl 可以通过 `exec` 调用 Shell 命令，例如用 `echo` 的重定向语法

```
1 exec echo "shell" > build/out.txt
```

当然，你可能需要先在 Shell 上通过 `mkdir build` 创建这个 `build` 目录。

最佳实践：**确保输出文件集中到某个特定目录，而不是和代码混在一起**。

Git

Git 是什么

Git 是一个版本控制软件，它以增量方式记录代码的修改，保存完整的版本历史。

你完全可以用 Git 管理课程服务器上的个人项目，不需要注册任何账号，不需要联网。

立志成才 报国裕民

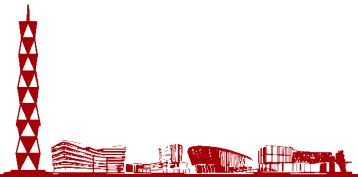
Git 不等于 GitHub

Git 并不等同于 GitHub!

- **Git** 是一个版本控制软件，它运行在你自己的电脑上，负责记录版本历史。
- **GitHub** 是一个很流行的代码托管平台，是一个网站，用于托管远程的 Git 仓库。

Git 和 GitHub 是两个不同层次的事物，GitHub 也只是众多 Git 仓库托管平台之一。

在本课程中，**我们只会使用 Git 最基础的功能实现对我们编写的 Tcl 脚本的版本控制**!



Git 能为我解决什么问题?

版本控制的意义在于，让你的每一次修改都会被记录，**想想以下的场景**

- 我改错代码了，现在一切都崩溃了，而我完全忘记几分钟前修改了什么。
- 我希望回退到昨天的那个版本，今天改完结果更差了，而我不知道如何回到上一版。
- 我在助教的建议下修改了几个配置，结果突然变好了，但到底是什么修改起效了？
- 我侥幸跑通了，交互式手敲了几条命令，但如果再来一次我还有把握得到同样的结果吗？
- 我很谨慎，但我复制出了五个项目目录：v1、v2、final、FINAL、FINALLLLL、NEW。

Git 配置

在首次提交前，需要先配置用户名和邮箱，它们会被记录在每一次提交中

```
git config --global user.name "Your Name"
git config --global user.email "you@example.com"
```

选项 `--global` 表示全局配置，对本机的所有仓库都生效。

立志成才 报国裕民

Git 创建仓库

Git 仓库（Repository）是 Git 用来管理一个项目的版本历史的核心结构。

若希望对某个项目使用 Git 版本控制，就要在该项目的目录下初始化 Git 仓库

```
1 git init
```

完成后，当前目录会出现一个 `.git` 隐藏目录，所有版本控制相关的数据都会存储在该目录中。

Git 提交更改

将改动添加到暂存区（Add），其中 . 表示当前目录下的所有改动

```
git add .
```

将改动提交到版本库（Commit），其中 -m 用于输入本次提交的说明

```
git commit -m "add tcl script"
```

就这样，你已经完成了一次最基本的**提交**！[3]

当然，你也可以在 VSCode 的图形界面中更为灵活地完成修改的暂存和提交。

[3]在 Git 的术语中，提交（Commit）就相当于是创建一个版本，这是一个纯粹的本地行为。

立志成才 报国裕民

Git 查看状态与历史

查看当前 Git 仓库的状态和所在分支（若没有任何改动，会提示“干净的工作区”）

```
git status
```

查看当前 Git 仓库的版本历史，选项 --graph 可以提供更漂亮的输出

```
git log
git log --graph
```

当然，你也可以在 VSCode 的图形界面中更直观地看到这些信息。

立志成才 报国裕民

Git 忽略文件

Git 只应当追踪源代码的变化，诸如编译结果或运行输出的“产物”绝不应该纳入版本控制！

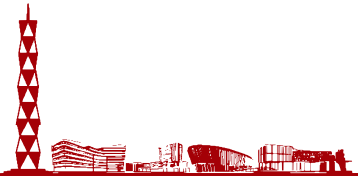
例如，若要忽略 `build` 目录，你应该创建一个 `.gitignore` 文件并写入

```
build/
```

Git 的设计基于这样一个假定：**只有源代码是关键的，构建产物是随时可以丢弃并重新产生的。**

数字后端适合用 Git 吗?

“遗憾的是，从软件工程的角度，数字后端工程师仍然在使用九十年代的开发方式”



数字后端适合用 Git 吗？

然而，在数字后端领域，以 Git 为代表的版本控制工具的普及度很低，有几个原因

- **习惯原因**：完整的 Tcl 脚本构成的 Flow 有专人维护，大部分工程师只会负责其中一个步骤的工具，针对不同设计做相应优化，更习惯使用交互式的 Tcl 命令，没有版本控制的概念。
- **现实原因**：在数字后端领域，构建产物的价值很高。对于一个大型的设计，综合流程和实现流程的耗时可能是以天甚至以星期计的。相比之下，它们对应的 Tcl 脚本反而没有什么实际价值。当工程师的协作是基于构建产物而不是它们编写的脚本，那 Git 就不再适用了。

但在课程学习中，我仍然认为使用 Git 对 Tcl 脚本进行版本控制是一个好的选择。课程中的设计规模都很小，完整重跑一次可能只要几分钟，对于初学者，确保结果的可复现性更为重要。

当然，对于这门课程而言，**Git 并不是强制性的内容，这只是为了能让你轻松一些**。

立志成才 报国裕民

Makefile

Makefile 是什么

Make 是一个构建自动化工具，Makefile 是其配置文件，它可以将构建命令固化为一条条规则。

朴素的想法：我记不住运行我脚本需要的那个很长很长的命令，我只是想让它**运行**！

立志成才 报国裕民

为什么使用 Makefile

例如，我们运行前面编写的 Tcl 脚本，要使用以下命令

```
tclsh main.tcl
```

同时，我们还需要确保 `build` 目录在运行 Tcl 脚本前已经被创建

```
mkdir build
```

当然这里 `tclsh main.tcl` 看起来还很简单，但这条命令之后可能会变得更复杂：工具可能需要带参数，脚本可能路径更深，试想下 `genus -batch -files script/genus/genus.tcl`！

Makefile 可以解决这些问题：**运行可以用一条简单命令包装，运行的前置条件可以自动被处理**。

立志成才 报国裕民

规则：目标、依赖、配方

Makefile 的基本结构是**规则**（Rule），你可以创建一个名为 `Makefile` 的文件

- **目标**：我最终要生成什么文件？（Target）
- **依赖**：目标的生成基于哪些文件？（Prerequisite）
- **配方**：目标是通过何种命令生成的？（Recipe）

```
target: prerequisite
    recipe
```

运行 `make target` 会让你构建 `target`，而直接运行 `make` 会默认构建第一个目标。

特别提醒：Makefile 中的缩进必须是 Tab 而不能是视觉上等效的四个 Space！

立志成才 报国裕民

规则：时间戳与按需构建

Makefile 利用文件的时间戳实现了按需构建

- 若目标不存在，则需要进行构建。
- 若目标存在，但时间戳早于某个依赖[4]，则也需要重新构建。

简单来说，如果目标的时间戳晚于它的所有依赖，那么目标必然就是最新的！

[4] 目标上一次被构建后，依赖又被修改了。

立志成才 报国裕民

Makefile 的基本用法

尝试创建一个名为 `Makefile` 的文件，写入

```
run:
    tclsh main.tcl
```

尝试运行

```
make run
```

现在，你再也不用关心你的解释器和你的脚本到底叫什么了，你只管 `make run` 运行！

立志成才 报国裕民

Makefile 的构建目录依赖

现在，我们来确保 build 目录会在运行前被自动创建

```
run: | build
    tclsh main.tcl

build:
    mkdir -p build
```

当你输入 make run 时，由于目标 run 依赖于 build，若 build 尚且不存在，就会通过下面的规则被 mkdir -p build 创建。这里 | 代表其后的依赖只需要存在即可，不关心时间戳先后。



Makefile 的构建目录移除

当然，我们可以添加一些新功能，例如包装一下 build 目录的移除

```
clean:
    rm -r -v -I build
```

尝试运行

```
make clean
```

其中，-r 代表递归删除，-v 代表列出删除内容，-I 代表单次确认。

立志成才 报国裕民

Makefile 的变量


```
BUILD_DIR:=build
TCL_MAIN:=main.tcl
TCLSH:=tclsh

.PHONY: run clean

run: | ${BUILD_DIR}
    ${TCLSH} ${TCL_MAIN}

clean:
    rm -r -v -I ${BUILD_DIR}

${BUILD_DIR}:
    mkdir -p $@
```

Makefile 的变量

Makefile 可以引入变量来更高效地组织规则

- 变量一般采用大写和下划线命名，例如 `BUILD_DIR`，这和 Tcl 的习惯类似。
- 变量在使用时要用 `${}` 包裹，但定义时不用，这和 Tcl 的习惯类似。
- 变量赋值一般用 `:=`，这是最符合通常认知的赋值，不要随意使用 `=` 赋值。
- `.PHONY: run clean` 用于声明**伪目标**，目标有两类，比如 `${BUILD_DIR}` 运行完成之后真的会生成一个名为 `build` 的目录，而 `run` 和 `clean` 运行后不会有对应文件产生，它们只是为了给 `make run` 和 `make clean` 提供入口，不是真正的目标，因此称之为伪目标。

立志成才 报国裕民

数字后端适合用 Makefile 吗?

数字后端领域对 Makefile 的使用事实上是畸形的，由于 EDA 工具往往会产生大量的日志和报告，我们很难厘清脚本产物和脚本之间的依赖关系，更多的只是用 `make run` 这样的伪目标来包装脚本运行所需的复杂命令。尽管如此，使用 Makefile 仍然能极大地减少我们的重复劳动。

当然，对于这门课程而言，**Makefile 并不是强制性的内容，这只是为了能让你轻松一些**。

项目组织

项目组织

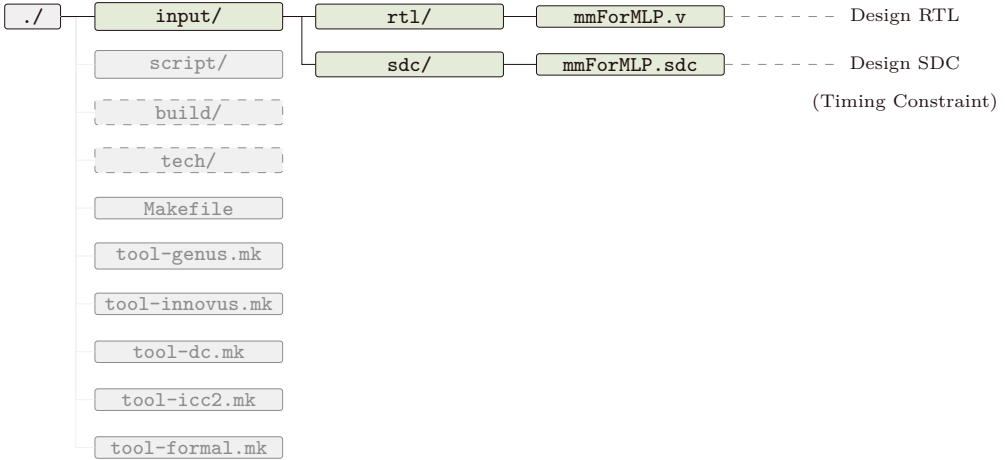
“不以规矩，不能成方圆”

立志成才 报国裕民

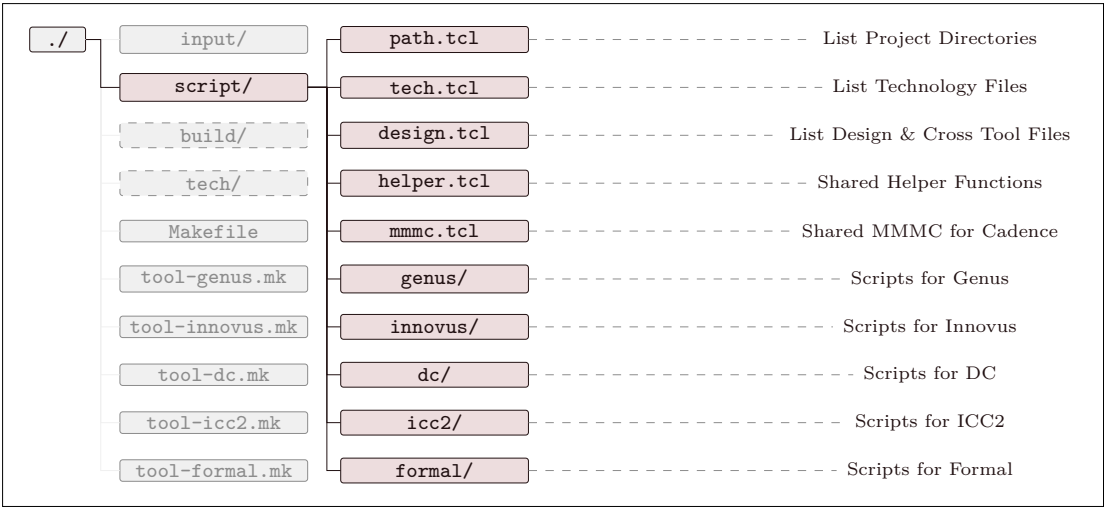
项目结构：./


./
input/
Design Input
script/
Tool Scripts
build/
Tool Output
tech/
Technology Files
Makefile
Makefile
tool-genus.mk
Makefile for Genus
tool-innovus.mk
Makefile for Innovus
tool-dc.mk
Makefile for DC
tool-icc2.mk
Makefile for ICC2
tool-formal.mk
Makefile for Formal

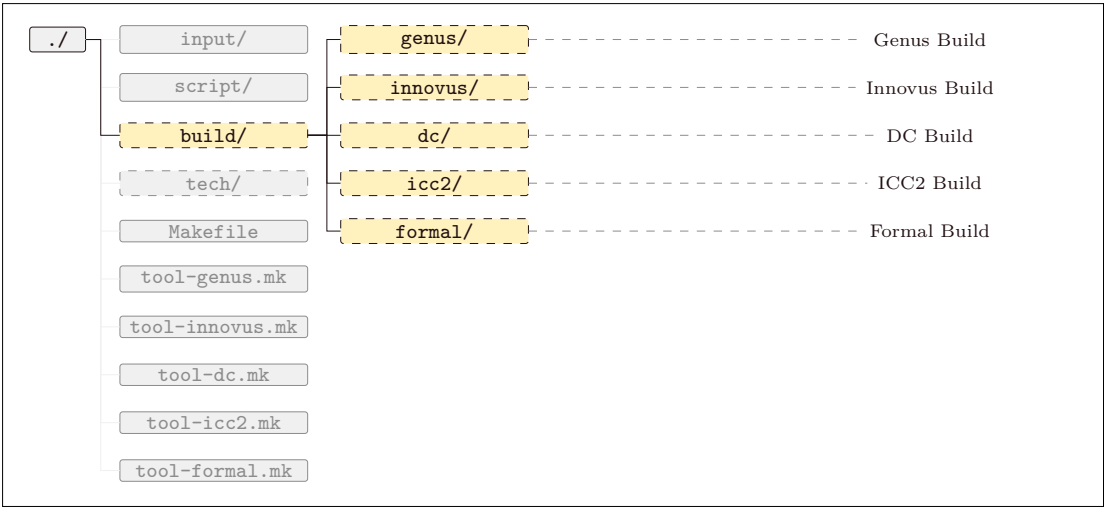
项目结构：input


./
input/
script/
build/
tech/
Makefile
tool-genus.mk
tool-innovus.mk
tool-dc.mk
tool-icc2.mk
tool-formal.mk
rtl/
mmForMLP.v
Design RTL
sdc/
mmForMLP.sdc
Design SDC
(Timing Constraint)

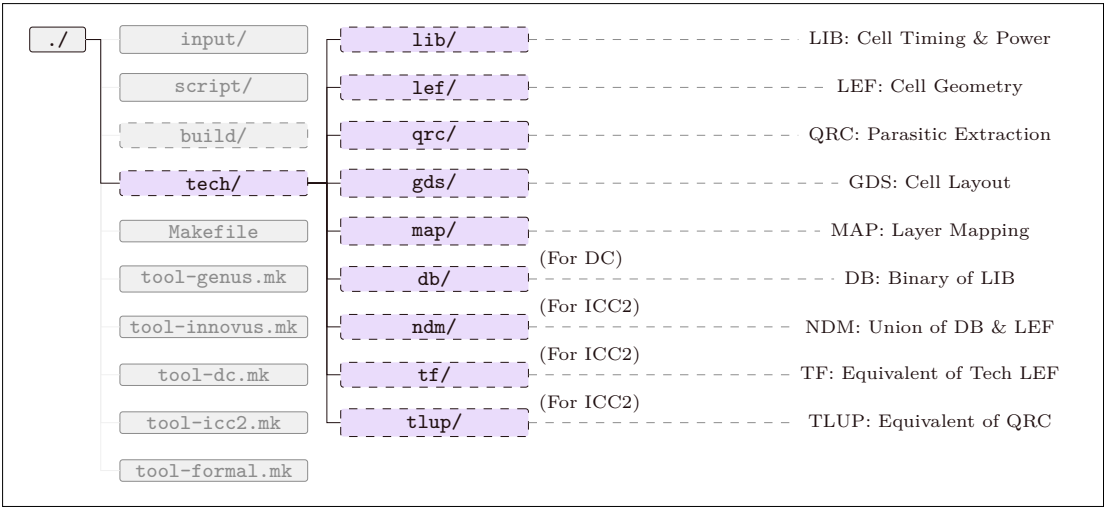
项目结构：script


./
input/
script/
build/
tech/
Makefile
tool-genus.mk
tool-innovus.mk
tool-dc.mk
tool-icc2.mk
tool-formal.mk
path.tcl
tech.tcl
design.tcl
helper.tcl
mmmc.tcl
genus/
innovus/
dc/
icc2/
formal/
List Project Directories
List Technology Files
List Design & Cross Tool Files
Shared Helper Functions
Shared MMMC for Cadence
Scripts for Genus
Scripts for Innovus
Scripts for DC
Scripts for ICC2
Scripts for Formal

项目结构：build


./
input/
script/
build/
tech/
Makefile
tool-genus.mk
tool-innovus.mk
tool-dc.mk
tool-icc2.mk
tool-formal.mk
genus/
innovus/
dc/
icc2/
formal/
Genus Build
Innovus Build
DC Build
ICC2 Build
Formal Build

项目结构：tech


./
input/
script/
build/
tech/
Makefile
tool-genus.mk
tool-innovus.mk
tool-dc.mk
tool-icc2.mk
tool-formal.mk
lib/
lef/
qrc/
gds/
map/
db/
ndm/
tf/
tlup/
LIB: Cell Timing & Power
LEF: Cell Geometry
QRC: Parasitic Extraction
GDS: Cell Layout
MAP: Layer Mapping
(For DC)
DB: Binary of LIB
(For ICC2)
NDM: Union of DB & LEF
(For ICC2)
TF: Equivalent of Tech LEF
(For ICC2)
TLUP: Equivalent of QRC

挂载工艺库

课程提供的工艺库与参考项目位于教师机的共享目录中，你需要手动完成挂载。

第一步，移除占据挂载点的错误本地软链接

```
sudo rm /mnt/nfs_share/EE291F
```

第二步，完成临时挂载（重启后会失效）

```
sudo mount -t nfs 10.15.27.50:/mnt/sharefile /mnt/nfs_share
```


立志成才 报国裕民

挂载工艺库

第三步，实现持久化的挂载，使重启后会自动完成挂载

```
sudo vim /etc/fstab
```

写入以下内容（按 i 进入插入模式，按 :wq 退出）

```
10.15.27.50:/mnt/sharefile /mnt/nfs_share nfs defaults,nfsvers=3,ro 0 0
```

复制参考项目

挂载成功后，可以将课程提供的参考项目完整复制一份到你的工作区用于对照

```
cd /home/workspace
cp -r /mnt/nfs_share/EE291F/3003/FlowLabMLP FlowLabMLP.ref
```

后续参考项目可能随着课程的推进逐渐完善。

从零建立自己的项目

在自己的工作区里新建项目目录，并从一开始就用 Git 管理

```
cd /home/workspace
mkdir FlowLabMLP
cd FlowLabMLP
git init
```

Makefile

定义项目名、构建目录、脚本目录，后续的规则会引用它们

```
PROJECT:=mmForMLP
BUILD_DIR:=build
SCRIPT_DIR:=script
```

实现构建目录的创建和清除

```
.PHONY: clean
```

```
clean:
    rm -r -v -I ${BUILD_DIR}
${BUILD_DIR}:
    mkdir -p $@
```

Makefile

定义 RUN_STAMP，用于获得时间戳（尝试运行 date +%Y%m%d-%H%M%S）

```
RUN_STAMP:=$(shell date +%Y%m%d-%H%M%S)
```

定义 GIT_COMMIT，用于获得版本好（尝试运行 git log -1 --pretty=tformat:"%H"）

```
GIT_COMMIT:=$(shell git log -1 --pretty=tformat:"%H")
```

定义 RUN_COMMAND，用于获得命令行实际输入的 make ... 命令

```
RUN_COMMAND:=make $(MAKECMDGOALS) $(MAKEOVERRIDES)
```

这三者结合起来，可以为后续工具的使用添加可审计可溯源的运行记录。

path.tcl

path.tcl 的职责是列出该项目结构定义下每个目录的路径。

首先，如何**足够可靠**的获取项目的根目录？（**不依赖工具的启动位置，不依赖硬编码的绝对路径**）

```
1 set PROJECT_DIR    [file dirname [file dirname [file normalize [info script]]]]
```

通过 [info script] 可以获取该脚本自身的路径，即 ./script/path.tcl，这是被项目结构所锚定的！随后，[file normalize] 获得绝对路径，[file dirname] 重复两次获得项目根目录。

列出所有一级目录

```
set TECH_DIR        [file join ${PROJECT_DIR} tech]
set INPUT_DIR       [file join ${PROJECT_DIR} input]
set BUILD_DIR       [file join ${PROJECT_DIR} build]
set SCRIPT_DIR      [file join ${PROJECT_DIR} script]
```

path.tcl

列出 tech/ 下的所有目录

```
set LEF_DIR         [file join ${TECH_DIR} lef]
set LIB_DIR         [file join ${TECH_DIR} lib]
set DB_DIR          [file join ${TECH_DIR} db]
set QRC_DIR         [file join ${TECH_DIR} qrc]
set GDS_DIR         [file join ${TECH_DIR} gds]
set MAP_DIR         [file join ${TECH_DIR} map]
set TF_DIR          [file join ${TECH_DIR} tf]
set NDM_DIR         [file join ${TECH_DIR} ndm]
```

列出 input/ 下的所有目录

```
1 set RTL_DIR        [file join ${INPUT_DIR} rtl]
2 set SDC_DIR        [file join ${INPUT_DIR} sdc]
```

path.tcl

列出 build/ 下的所有目录（各工具的构建目录）

```
set GENUS_BUILD    [file join ${BUILD_DIR} genus]
set DC_BUILD       [file join ${BUILD_DIR} dc]
set INNOVUS_BUILD  [file join ${BUILD_DIR} innovus]
set ICC2_BUILD     [file join ${BUILD_DIR} icc2]
set FORMAL_BUILD   [file join ${BUILD_DIR} formal]
```

path.tcl

列出 script/ 下的所有目录（各工具的脚本目录）

```
set GENUS_SCRIPT   [file join ${SCRIPT_DIR} genus]
set DC_SCRIPT      [file join ${SCRIPT_DIR} dc]
set INNOVUS_SCRIPT [file join ${SCRIPT_DIR} innovus]
set ICC2_SCRIPT    [file join ${SCRIPT_DIR} icc2]
set FORMAL_SCRIPT  [file join ${SCRIPT_DIR} formal]
```

design.tcl

design.tcl 的职责是列出该项目的设计文件和设计相关名称，后续也会包含各工具的交付。

```
set DESIGN          mmForMLP
set CLK_NAME        main_clk
set PORT_CLK        clk
set PORT_RST        resetn
set RTL_LIST        [list [file join ${RTL_DIR} ${DESIGN}.v]]
set SDC_FILE        [file join ${SDC_DIR} ${DESIGN}.sdc]
```

其中，CLK_NAME 是 SDC 中定义的时钟名，PORT_CLK 和 PORT_RST 是 RTL 中的网表名。

Thank You

谢谢观看