

| STAT243 Lecture 3.4 Bash Shell-Shell Programming

| 1 Shell Scripts

| 1.1 Overview

Shell scripts 是包含一系列 shell 命令的文本文件（通常以 `.sh` 结尾）。通过编写脚本，你可以自动化一组命令的执行，而不必在终端中手动输入每一行。

| 1.1.1 运行 Shell 脚本的方式

| 1.1.1.1 方法 1：使用 `source` 或 `.`

```
>- Shell
1 $ source ./file.sh
2 # 或
3 $ . ./file.sh
```

- `source`（或 `.`）在**当前 shell 环境中**执行脚本内容；
- 脚本中定义的变量和函数会保留在当前会话中。

| 1.1.1.2 方法 2：直接运行脚本

如果直接输入脚本名（如 `file.sh`），可能遇到以下问题：

1. **找不到文件**（脚本未在 `$PATH` 路径中）；
2. **缺乏执行权限**（未设置可执行标志 `-x`）。

正确方式：

1. 在脚本开头声明解释器（shebang）：

```
>- Shell
1 #!/bin/bash
```

告诉系统使用 Bash 来解释执行脚本。

2. 赋予可执行权限：

```
>- Shell
1 $ chmod +x file.sh
```

3. 执行脚本：

```
>- Shell
1 $ ./file.sh
```

🔗 Logic ▾

建议始终在脚本首行加上 `#!/bin/bash`，确保脚本在不同系统中使用正确的解释器。

| 2 Functions

2.1 Overview

Bash 函数是将多条命令封装在一起的可复用单元，相比 alias 更强大，支持参数传递与逻辑控制。

定义语法：

```
>- Shell
1 function name() {
2     commands
3 }
```

调用方式：

```
>- Shell
1 $ name arg1 arg2 ...
```

2.2 参数传递

在函数中，Bash 会自动创建以下特殊变量：

变量	含义
\$1, \$2, \$3, ...	依次表示第 1、2、3 个参数
\$#	参数个数
\$@	所有参数（以空格分隔）

2.2.1 Example: 自定义上传函数

```
>- Shell
1 function putscf() {
2     scp $1 jarrod@arwen.berkeley.edu:$2
3 }
```

执行：

```
>- Shell
1 $ putscf unit1.pdf teaching/243/.
```

该命令将 unit1.pdf 上传到远程服务器目录 ~/teaching/243/。

🔗 Logic ▾

函数可放入 .bashrc 文件中，实现登录后自动加载。

3 If / Then / Else

3.1 控制流结构

Shell 支持 条件分支 (if-then-else) 语法：

```
>- Shell
```

```

1  if [ condition ]; then
2      commands
3  elif [ other_condition ]; then
4      commands
5  else
6      commands
7  fi

```

- 条件判断语句**必须用方括号包裹** (`[]`),
- 每个部分之间**需要空格**,
- 结尾**必须用 `fi` 结束**。

3.2 Example: `niceR` 函数

```

>- Shell
1  # niceR: 提交优先级较低的 R 作业
2  # 用法: niceR inputRfile outputRfile
3  # Author: Brian Caffo
4
5  function niceR() {
6      if [ $# != "2" ]; then
7          echo "usage: niceR inputRfile outputfile"
8      elif [ -e "$2" ]; then
9          echo "$2 exists, I won't overwrite"
10     elif [ ! -e "$1" ]; then
11         echo "inputRfile $1 does not exist"
12     else
13         echo "running R on $1"
14         nice -n 19 R --no-save < $1 &> $2
15     fi
16 }

```

- `-e` 检查文件是否存在;
- `! -e` 检查文件是否不存在;
- `nice -n 19` 表示降低进程优先级, 以免占用过多 CPU;
- `&>` 同时重定向 stdout 与 stderr。

⚠ Remark ▾

将 `then` 放在单独一行可省略分号;
即:

```

>- Shell
1  if [ condition ]
2  then
3      ...
4  fi

```

3.3 Example: 字符串比较

```

>- Shell

```

```

1  var="some text"
2
3  if [ "${var}" == "some text" ]; then
4      echo "found equal"
5  fi
6
7  if [ "${var}" != "some text" ]; then
8      echo "found not equal"
9  fi

```

- 建议始终为变量加双引号，避免空格或通配符误解析。

⚠ Remark ▾

在 Bash 中，字符串比较时若省略引号，
空格与 `*` 等符号可能被误认为分隔符或通配符，导致逻辑错误。

| 4 For Loops

| 4.1 Overview

在 Bash 中，**for 循环** 用于遍历一组文件、目录或变量值。
它是 shell 脚本中最常用的控制结构之一，常用于批量重命名、下载文件、或启动任务。

| 4.2 Example 1: 文件批量重命名

```

>- Shell
1  $ for FILE in $(ls *.txt); do
2  >     mv $FILE ${FILE/.txt/.R}
3  >     # 将文件扩展名 .txt 替换为 .R
4  > done

```

🔗 Logic ▾

- `${FILE/.txt/.R}` 是 **参数替换语法**，将变量内容中的 `.txt` 替换为 `.R`；
- `>` 提示符表示 Shell 等待多行输入（循环尚未结束）。

| 4.3 Example 2: 自动化文件下载

```

>- Shell
1  # forloopDownload.sh
2  # 使用 wget 下载多个文件
3  # Author: Chris Paciorek
4  # Date: July 28, 2011
5
6  url='ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/daily/grid/years'
7  types="tmin tmax"
8
9  for ((yr=1950; yr<=2017; yr++))
10 do
11     for type in ${types}
12     do
13         wget ${url}/${yr}.${type}
14     done
15 done

```

说明：

- 双层循环：外层遍历年份，内层遍历文件类型；
- `for ((...))` → 数值循环语法（C 风格）；
- `wget` 从 FTP 服务器批量下载文件。

Logic ▾

若 `do` 独占一行，可省略分号。

即：

```
>- Shell
1  for item in list
2  do
3      ...
4  done
```

4.4 Example 3: 批量启动任务



Shell

```
1  # forloopJobs.sh
2  # 启动一系列 R 模拟任务
3  # Author: Chris Paciorek
4  # Date: July 28, 2011
5
6  n=100
7  for (( it=1; it<=100; it++ ))
8  do
9      echo "n=$n; it=$it; source('base.R')" > tmp-$n-$it.R # 创建定制 R 文件
10     R CMD BATCH --no-save tmp-$n-$it.R sim-n$n-it$it.Rout # 执行任务
11 done
12 # 注意 base.R 不应在脚本中定义 n 或 it
```

⚠ Remark ▾

若任务间仅参数不同，更好的做法是通过 `commandArgs()` 或环境变量在 R 脚本中读取参数，而非生成多个临时文件。

4.5 Example 4: 自定义分隔符循环

默认情况下，`for` 循环以 `空格` 为分隔符。

若要更改分隔符，可修改变量 `IFS`（Internal Field Separator）。



Shell

```
1  $ IFS=:
2  $ types=tmin:tmax:pmin:pmax
3  $ for type in $types
4  > do
5  >     echo $type
6  > done
```

输出：

```
1 tmin
2 tmax
3 pmin
4 pmax
```

Logic ▾

IFS 控制 Bash 如何拆分字符串。
该技巧常用于解析 CSV、路径或自定义格式字符串。