# Shell脚本代码规范

黄嘉桢 2022141461075

软件过程管理下半期作业

## **一、基础规范**

### 1. Shebang声明

每个脚本必须以#!/usr/bin/env bash或#!/bin/sh开头。前者用于需要bash特性的脚本，后者用于追求最大兼容性的场景。

### 2. 文件格式

脚本文件必须使用Unix换行符(LF)，避免Windows的CRLF格式导致执行错误。可在.gitattributes中配置\*.sh text eol=lf确保团队协作一致性。

### 3. 文件扩展名

源文件建议使用.sh扩展名，便于编辑器识别并启用语法高亮。

### 4. 缩进规范

使用2个空格缩进，禁用Tab。这保证了在不同编辑器和终端中显示一致。

### 5. 行长限制

每行不超过80字符。长管道命令应在|后换行并缩进：

grep 'ERROR' /var/log/app.log | \  
 awk '{print $1, $5}' | \  
 sort | \  
 uniq -c

### 6. 严格模式

脚本开头必须启用set -euo pipefail：

* -e: 命令失败立即退出
* -u: 使用未定义变量时报错
* -o pipefail: 管道中任何命令失败都返回错误

## **二、变量管理**

### 7. 局部变量声明

函数内变量必须用local声明，避免污染全局作用域：

process\_file() {  
 local file\_path="$1"  
 local result  
 # ...  
}

### 8. 变量命名规范

* 环境变量和常量：UPPERCASE\_WITH\_UNDERSCORES
* 局部变量：lowercase\_with\_underscores
* 避免单字母变量名（循环计数器除外）

### 9. 变量引用

必须用双引号引用变量"$var"，防止空格分割和通配符展开问题。

### 10. 花括号使用

使用${var}明确变量边界，特别是字符串拼接时：

echo "I like ${fruit}s" # 而不是 $fruits

### 11. 数组遍历

使用"${array[@]}"确保元素中的空格被正确处理：

files=("file one.txt" "file two.txt")  
for file in "${files[@]}"; do  
 echo "Processing: $file"  
done

### 12. 命令替换

使用$(command)而非反引号，支持嵌套且更清晰：

current\_time=$(date +%Y-%m-%d)

## **三、函数与控制流**

### 13. 主函数模式

将主逻辑封装在main()函数中，脚本末尾调用：

main() {  
 # 主逻辑  
}  
main "$@"

### 14. 函数定义

使用POSIX兼容的语法：

my\_function() {  
 # 函数体  
}

### 15. 条件测试

使用[[ ]]进行条件测试，功能更强大：

if [[ "$user" == "admin" && -n "$password" ]]; then  
 # ...  
fi

### 16. 文件迭代

禁止解析ls输出，使用find或glob模式：

# 推荐  
for log in /var/log/\*.log; do  
 [[ -e "$log" ]] || continue  
 process\_log "$log"  
done

### 17. Case语句格式

每个分支缩进2空格，以;;结束：

case "$action" in  
 start)  
 start\_service  
 ;;  
 stop)  
 stop\_service  
 ;;  
 \*)  
 echo "Unknown action" >&2  
 exit 1  
 ;;  
esac

## **四、错误处理与调试**

### 18. 临时文件清理

使用trap确保清理：

temp\_file=$(mktemp)  
trap 'rm -f "$temp\_file"' EXIT INT TERM

### 19. 错误信息输出

错误信息必须重定向到stderr：

echo "Error: File not found" >&2

### 20. 调试模式

通过环境变量控制调试：

[[ "${DEBUG:-0}" == "1" ]] && set -x

### 21. 命令失败处理

对预期可能失败的命令显式处理：

if ! command -v docker &>/dev/null; then  
 echo "Docker is not installed" >&2  
 exit 1  
fi

## **五、系统管理最佳实践**

### 22. 使用printf而非echo

printf行为更可预测，特别是处理特殊字符时：

printf "%s\n" "$user\_input"

### 23. 避免eval

除非绝对必要，禁止使用eval，防止命令注入。

### 24. 安全创建临时文件

必须用mktemp创建临时文件：

temp\_dir=$(mktemp -d) || exit 1

### 25. 参数传递

使用"$@"传递所有参数，保持参数完整性：

wrapper\_function() {  
 original\_command "$@"  
}

## **六、日志与监控**

### 26. 结构化日志

实现带时间戳和级别的日志函数：

log() {  
 local level="$1"  
 local message="$2"  
 printf '[%s] %s: %s\n' "$(date -Iseconds)" "$level" "$message"  
}

### 27. 系统日志集成

重要事件发送到syslog：

logger -t my\_script -p user.info "Backup completed successfully"

### 28. 敏感信息保护

禁止在日志中记录密码、API密钥等敏感信息。

## **七、跨平台兼容**

### 29. 路径转换(WSL)

在WSL中使用wslpath转换路径：

win\_path="C:\\Users\\Data"  
linux\_path=$(wslpath -u "$win\_path")

### 30. 路径转换(Git Bash)

在Git Bash中使用cygpath：

win\_path="D:\\projects"  
bash\_path=$(cygpath -u "$win\_path")

### 31. 命令存在性检查

使用前检查命令是否可用：

if ! command -v jq &>/dev/null; then  
 echo "Please install jq first" >&2  
 exit 1  
fi

## **八、权限与安全**

### 32. 最小权限原则

只对必要命令使用sudo，而非整个脚本：

# 脚本内部  
sudo systemctl restart nginx

### 33. 文件权限检查

操作前验证权限：

if [[ ! -w "$log\_file" ]]; then  
 echo "Cannot write to log file" >&2  
 exit 1  
fi

### 34. 输入验证

对用户输入进行验证：

if [[ ! "$port" =~ ^[0-9]+$ ]]; then  
 echo "Invalid port number" >&2  
 exit 1  
fi

## **九、文档与注释**

### 35. 脚本头部注释

每个脚本必须包含用途说明：

#!/usr/bin/env bash  
# Purpose: Backup MySQL databases  
# Usage: ./backup.sh <database\_name>  
# Dependencies: mysqldump, gzip

### 36. 函数文档

重要函数需要说明参数和返回值：

# Delete old log files  
# Arguments:  
# $1: Directory path  
# $2: Days to keep  
# Returns:  
# 0 on success, 1 on error  
cleanup\_logs() {  
 # ...  
}

### 37. TODO标记

未完成功能标记责任人：

# TODO(john): Add error recovery logic

## 十、高级技巧

### 38. 进程管理

使用作业控制管理后台任务：

command &  
pid=$!  
# ... 其他操作 ...  
wait $pid || echo "Command failed"

### 39. 信号处理

合理处理中断信号：

trap 'echo "Interrupted"; cleanup; exit 130' INT

### 40. 锁文件机制

防止脚本重复运行：

lock\_file="/var/run/myscript.lock"  
exec 200>"$lock\_file"  
if ! flock -n 200; then  
 echo "Script already running" >&2  
 exit 1  
fi

### 41. 超时控制

为长时间运行的命令设置超时：

timeout 30s some\_long\_command || {  
 echo "Command timed out" >&2  
 exit 1  
}

### 42. 配置文件加载

安全加载配置：

config\_file="/etc/myapp/config"  
[[ -r "$config\_file" ]] && source "$config\_file"