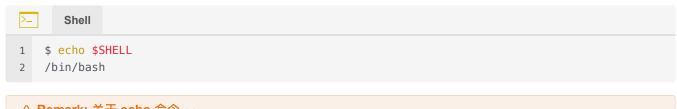
### STAT243 Lecture 3.1 Bash Shell-Overview

# |1 访问 Shell

# |1.1 查看默认 Shell



## ⚠ Remark: 关于 echo 命令 ∨

echo 是一个 Bash 命令, 用于在终端显示文本或变量的值

### I echo 的主要用途

• 显示简单的文本: 可以用 echo 来打印任何字符串。

Shell

1 echo "Hello, world!"

输出:

1 Hello, world!

• 显示变量的值: 在脚本中, echo 通常用来检查变量是否被正确设置

Shell

1 name="Alice"
2 echo "My name is \$name."

输出:

1 My name is Alice.

- 输出到文件: 你可以使用重定向操作符 ( > 或 >> )将 echo 的输出写入文件,而不是显示在终端。
  - > 会覆盖文件原有内容。
  - >> 会将新内容追加到文件末尾。

1 echo "This is the first line." > file.txt
2 echo "This is the second line." >> file.txt

运行后, file.txt 文件内容为:

- 1 This is the first line.
- 2 This is the second line.

### lecho 的常见选项

• -n: 不在输出的末尾添加换行符, 可以在同一行上连续输出内容

```
1 echo -n "Starting..."
2 echo " Done."
输出:
```

# |1.2 切换到 Bash (one-time basis)

```
Shell

1 $ bash
```

# 1.3 设置为默认

```
Shell

1 $ chsh /bin/bash
```

其中的路径应该为 bash shell 的路径, 可以通过以下方法确定:

```
$ Shell

1  $ type bash
2  bash is /usr/bin/bash
```

# |2 使用变量

## |2.1 访问和打印变量

- 通过在变量前添加 🕏 访问变量
- 通过 echo 命令来打印变量

```
Shell

1  $ echo $USER
2  paciorek
```

# |2.2 声明变量

声明变量时无需使用 \$

```
Shell

1 $ counter=1
```

- 1. 由于 bash 使用空格来解析你输入的表达式, 因此需要确保等号周围没有空格 (否则 bash 会将 counter 视作一个命令)
- 2. 可以将变量名用花括号括起来, 以确保 shell 知道变量名在哪里结束

### |2.3 环境变量

环境变量是一类特殊的 Shell 变量, 有助于控制 Shell 的 behavior, 通常以全大写命名

### | 2.3.1 查看环境变量

```
Shell

1 $ printenv
```

### 12.3.2 创建环境变量

```
Shell

1 $ export base=/home/jarrod/
```

- 不使用 export: 设置的变量只会在当前 Shell 内生效
- 使用 export: 设置的变量会在当前 Shell 及其派生 shell (如运行程序) 中生效
- 在 .bashrc 文件中使用 export:设置的变量始终生效

# I 2.3.3 控制 Bash prompt 的外观

可以通过对 PS1 进行以下修改来显示 username, hostname, 和 current working directory

#### 13 Introduction to Commands

### 3.1 Elements of a Command

- 一般而言, 命令行由 四个部分 组成:
  - 1. 命令 (command)
  - 2. 选项 (options)
  - 3. 参数 (arguments)
  - 4. <mark>执行确认 (line acceptance) ——</mark> 按 Enter 键执行

示例:

```
Shell

1 $ ls -l file.txt
```

- ls → 命令
- -1 → 选项 (long format, 长格式显示)
- file.txt → 参数(指定文件)
- 按下 Enter 执行命令

#### 3.1.1 Command Execution Process

当用户在 bash 提示符下按 Enter 后, 系统会:

- 1. 解析命令 (parse)
  - bash 会识别命令及其参数。
- 2. 查找命令来源
  - 先检查该命令是否为 shell function (自定义函数);
  - 若不是, 再查找是否为 shell builtin (内建命令);
  - 若两者都不是,则在 PATH 变量 指定的目录中依次查找可执行文件。

```
$ Shell

1  $ echo $PATH
2  /home/jarrod/usr/bin:/usr/local/bin:/usr/bin:
```

### | 3.1.2 Example: Command Lookup

```
Shell

1 $ grep pdf file.txt
```

#### 执行步骤:

- 1. 检查 grep 是否是函数或内建命令。
- 2. 若不是,则 bash 会在 \$PATH 路径列表中依次查找:
  - /home/jarrod/usr/bin
  - /usr/local/bin
  - /bin
  - /usr/bin
- 3. 一旦找到匹配的可执行文件,就运行它。

使用 type 可查看命令位置:

```
$ $ type grep
2 grep is hashed (/usr/bin/grep)
```

#### **| 3.1.3 Variable Substitution**

bash 会在执行命令前进行变量替换:

```
5 Shell

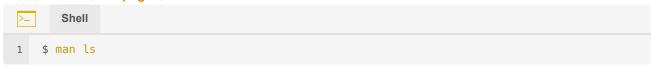
1  $ myfile=file.txt
2  $ grep pdf $myfile
```

执行时会被解析为:



## 3.2 Getting Help with Commands

• 查看命令手册 (man pages)



- man (manual) 命令显示系统自带帮助文档。
- 使用 q 退出手册界面。
- 使用帮助选项(--help)

```
Shell

1 $ ls --help
```

• 直接在终端输出简要帮助与可用选项。

## |4 Command Line 中的高效操作

### | 4.1 Tab 补全

在 Shell 中输入命令或文件名时, 可以按下 Tab 键, Shell 会尝试补全我们正在输入的命令或文件名:

- 如果搜索路径中只有一个匹配的命令, 那么 shell 会显示其值, 光标会停在补全名称之后一个空格处
- 如果有多个命令与部分名称匹配, shell 会尽可能多地补全; 此时连续按两次 Tab 键会显示一个选项列表, 并重新显示部分命令行以供进一步编辑

## 4.2 键盘快捷键

Key Strokes	Descriptions
Ctrl-a	Beginning of line 行首
Ctrl-e	End of line 行尾
Ctrl-k	Delete line from cursor forward 从光标处删除到行尾
Ctrl-w	Delete word before cursor 删除光标前的单词
Ctrl-y	pastes in whatever was deleted previously with Ctrl-k or Ctrl-w 粘贴之前用 Ctrl-k 或 Ctrl-w 删除的内容
ESC-F	Forward one word 向前移动一个单词
ESC-B	Backwards one word 向后移动一个单词
Ctrl-d	EOF; exit 退出
Ctrl-c	Interrupt current command 中断当前命令
Ctrl-z	Suspend current command 挂起当前命令

Key Strokes	Descriptions
Ctrl-l	Clear screen 清除屏幕
Ctrl-r	Enables an <u>interactive search history</u> 启用交互式搜索历史

# | 4.3 Command History and Editing

### 14.3.1 Navigating and Reusing Commands

- 使用 ↑/ ↓ 键浏览先前输入的命令。
  - 可直接按 Enter 重新运行;
  - 或修改后再执行。
- 使用 history 查看完整命令记录:

• 历史记录行为由以下环境变量控制:

```
Shell

1 $ echo $HISTFILE # 历史记录文件
2 $ echo $HISTSIZE # 记录的命令条数
```

# **| 4.3.2 Recalling Commands Quickly**

• 通过命令编号或字符串回调:

可在结尾加上:p 只打印、不执行:

命令	说明
11	上一条命令
!n	第 n 条命令
!-n	倒数第 n 条命令
!string	最后以 string 开头的命令
!?string	最后包含 string 的命令
^old^new	替换上一条命令中字符串 old → new

#### 示例:

```
      Shell

      1 $!-2 # 执行倒数第二条命令

      2 $!gi # 执行最后以 "gi" 开头的命令

      む Logic >
```

```
      2-
      Shell

      1 $!gi:p

      这样可以查看命令内容,再按↑键编辑或执行。
```

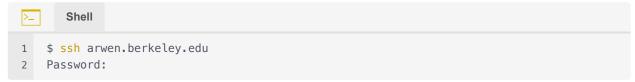
### **| 4.3.3 Searching Command History**

- Ctrl + r: 在命令历史中反向搜索匹配字符串。
  - 按 Enter 执行匹配命令。
  - 按 Ctrl + c 退出搜索模式。
  - 按 Esc 将搜索结果放回命令行以便编辑。

## **| 5 Accessing Remote Machines**

### 15.1 SSH: Secure Shell

- SSH 提供加密的远程登录方式(Linux / Mac 常用)。
  - 基本用法:



• 若远程用户名 (如 jarrod) 与本地机器上的用户名不同:

```
Shell

1  $ ssh jarrod@arwen.berkeley.edu
```

• 若需运行图形界面程序:

```
Shell

sh
```

### 

-X 参数启用 X11 forwarding,

允许在远程服务器运行的 GUI 程序显示在本地屏幕上。

## | 5.2 SCP: Secure Copy

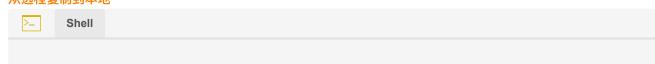
• 从本地复制文件到远程主机

```
Shell

1 $ scp file1.txt jarrod@arwen.berkeley.edu:.
```

将本地文件复制到远程用户主目录。

- 表示保持原文件名。
- 从远程复制到本地



- 1 \$ scp jarrod@arwen.berkeley.edu:file2.txt .
- 从一个远程主机复制到另一个远程主机

>\_ Shell

1 \$ scp jarrod@arwen.berkeley.edu:file3.txt jmillman@scf-ug02.berkeley.edu:.

• 复制整个目录

>\_ Shell

1 \$ scp -r src jmillman@arwen.berkeley.edu:.

-r 表示递归复制整个目录结构。

### 

scp 在底层仍通过 SSH 传输数据,因此具有同样的加密与安全性。

## 5.3 Using sudo for Administrative Access

- 某些操作(安装软件、修改系统设置)需要 root 权限。
- 在 Ubuntu 或 WSL 中,可使用 sudo 临时以管理员身份执行命令。

### 示例:

Shell

s sudo apt-get upgrade # 升级全部软件
sudo apt-get install vim # 安装 vim 编辑器

### **♦ Logic** ∨

sudo 代表 superuser do,

它让普通用户在授权情况下执行管理员操作。

Windows 用户可通过 Ubuntu Subsystem (WSL) 体验完整 Linux 环境并拥有 root 权限。