STAT243 Lecture 1 Unix Intro

1 Introduction to UNIX, Computers, and Key Tools

> **Logic**
>
> 本章目标：熟悉 UNIX 命令行、版本控制、计算机硬件基本组成、远程连接与常用文本编辑器，为后续编程与数据计算打下统一的环境与工作流基础。

1.1 UNIX command line basics（命令行基础）

> **Logic**
>
> 大多数科学计算在 UNIX 系统（Linux / macOS）上进行。命令行提供可组合、可追踪、可自动化的工作方式。

- **UNIX** 指类 UNIX 系统（Linux、macOS）。科研计算常通过 **SSH** 登录到远程 UNIX 服务器。
- 建议尽快完成入门教程（如 UNIX 基础、Shell scripting），确保能独立完成基本文件操作与脚本运行。

1.1.1 常用概念

- **Shell**：如 `bash`、`zsh`，提供命令解析与脚本执行。
- **Path**：`/` 为根目录；`~` 为用户 home；`.` 当前目录；`..` 上级目录。
- **权限**：`rwx`，可用 `chmod` / `chown` 调整。
- **I/O 重定向**：`>` 覆盖输出、`>>` 追加输出、`<` 输入重定向、`|` 管道。

1.1.2 常见操作示例

Shell

```shell
## 查看路径与文件
pwd
ls -lha

## 创建、移动、复制、删除
mkdir -p project/src
mv a.txt project/
cp -r project project_bak
rm -rf project_bak

## 查看内容与搜索
cat file.txt
less file.txt
grep -n "pattern" file.txt

## 环境信息
uname -a
which python
```

> **Remark**
>
> 在 Windows 环境进行科研计算时，建议使用 WSL（Windows Subsystem for Linux）或直接在远程 Linux 服务器上工作，以获得一致的 UNIX 体验。

1.2 Version control（版本控制：Git）

Logic

版本控制是现代科研与协作的核心。Git 基于逐行 diff 跟踪文本文件（代码、配置、文档）的演化。

- **Repository（仓库）**：项目文件与历史记录的集合。
- **Remote（远端）**：如 GitHub/GitLab；本地与远端通过 `fetch` / `push` 同步。

1.2.1 最小工作流

Shell

```shell
## 1) 克隆课程仓库（示例）
git clone https://github.com/berkeley-stat243/fall-2025

## 2) 同步远端更新
cd fall-2025
git pull

## 3) 本地开发常用
git status
git add <files>
git commit -m "feat: add notes for unit1"

## 4) 推送到远端（需提前设置权限）
git push origin main
```

Remark

GUI 客户端（如 GitHub Desktop）可降低上手难度，但必须理解基础命令与分支/合并概念，以便在冲突与回滚时自救。

1.3 Parts of a computer（计算机组成与性能瓶颈）

Logic

编程效率不仅取决于算法，也受硬件层次结构影响：CPU、缓存、内存与磁盘 I/O 的速度差异可能决定整体性能瓶颈。

- **CPU**：负责算术与逻辑运算；具备多级 **cache**（L1/L2/L3）。
- **Main memory（RAM）**：主存，容量中等，延迟低于磁盘但高于 cache。
- **Bus**：在 CPU、内存、外设之间传输数据的总线。
- **Disk（持久化存储）**：HDD/SSD，容量大但访问最慢；I/O 可能成为瓶颈。

1.3.1 典型瓶颈判断

- **Compute-bound**：CPU 算力成瓶颈。
- **Memory-bound**：内存带宽/访问延迟成瓶颈。
- **I/O-bound**：磁盘读写成瓶颈，需优化数据布局、批量 I/O、缓存与并行。

1.4 Connecting to other machines（远程连接与文件传输）

Logic

通过 SSH 进行远程登录与端口转发，通过 SCP/SFTP 可靠地搬运数据。

1.4.1 SSH 登录

Shell

```shell
## 登录到远程主机（示例用户名与主机）
ssh username@radagast.berkeley.edu

## 指定端口、开启压缩
ssh -p 22 -C username@host

## 首次连接会提示保存主机指纹；建议配置 SSH key 免密登录
```

1.4.2 SCP 文件传输

Shell

```shell
## 本地 → 远程
scp file.txt username@radagast.berkeley.edu:~/research/

## 远程 → 本地（重命名保存）
scp username@radagast.berkeley.edu:/data/file.txt \
    ~/research/renamed.txt
```

Remark

- 远程路径通常使用**绝对路径**或以 `~` 为基准的相对路径。
- 大量文件或断点续传场景，考虑 `rsync -avP` 以降低传输成本并便于续传。
- 若需图形界面，可选 SFTP 客户端（如 FileZilla、WinSCP）。

1.5 Editors（文本编辑器与 IDE）

Logic

科学计算需要**文本编辑器**，而非 Word 类富文本处理器。编辑器需良好支持代码、高亮、Lint、LSP 与可扩展性。

1.5.1 选择建议

- 传统 UNIX 编辑器：**emacs**, **vim**。
- 现代通用编辑器：**VS Code**（IDE 级别，支持 Python / R / Jupyter / Quarto，集成 Git 与调试；可用 GitHub Copilot）。
- 其他：Sublime Text（专有），Aquamacs Emacs（macOS），Notepad++（Windows），TextMate（macOS）。
- **RStudio**：R / Quarto / R Markdown 生态友好，亦可运行 Python 代码块。

Remark

- 不要用 Microsoft Word 或 Google Docs 编辑代码或 Markdown/Quarto/LaTeX。

- Windows 默认隐藏文件后缀，可能导致脚本扩展名判断错误，建议开启显示后缀。

1.5.2 (Optional) Basic emacs（快速上手）

- **Major modes**：针对不同文件类型启用特定编辑体验（Python、R、C、LaTeX 等）。
- **在终端中运行**：图形转发不便时可用 `emacs -nw file.txt`。
- **与解释器联动**：R 可用 ESS（Emacs Speaks Statistics）。

1.5.2.1 常用键位（部分）

- `C-x C-c` 退出；`C-x C-s` 保存；`C-x C-w` 另存。
- `C-s` 搜索；`ESC` 退出命令区。
- `C-a` 行首；`C-e` 行尾；`C-k` 剪切到行尾；`C-y` 粘贴（yank）。
- 选区：`C-space` 起始 → 移动光标到末尾 → 操作。

Example

若在 `git commit` 时默认打开 emacs 作为编辑器，输入 commit message 保存并退出：`C-x C-s` → `C-x C-c`。

1.5.3 (Optional) Basic vim（模式编辑器）

- 两种模式：**Normal**（导航/命令）与 **Insert**（插入文本）。
- 由 Normal 进入 Insert：按 `i`；退出回 Normal：`Esc`。
- 保存/退出/搜索：`:w`、`:x`、`:q`；`/pattern` 搜索；`n` / `N` 导航匹配。

Example

`git commit` 未加 `-m` 时默认进入 vim：按 `i` 进入 Insert，写 message；按 `Esc` 回 Normal，输入 `:wq` 保存退出。

2 Computer Architecture

Logic

目的：为后续 Shell、编程、性能优化提供硬件与系统层面的上下文。聚焦计算机的关键部件、程序如何运行、文件系统与操作系统抽象。

2.1 Components of Computer Architecture（部件一览）

Logic

以数据在各层之间流动为主线：CPU ↔ Cache ↔ RAM ↔ Storage，经由 Bus 与 Ports 连接外设。

2.1.1 CPU（Central Processing Unit）

- 职责：执行 **arithmetic / logic / control / I/O** 指令，是计算机的“控制中心”。
- 典型组成：

    - **Control Unit**：负责取指、译码与控制执行流程。
    - **ALU（Arithmetic Logic Unit）**：算术、逻辑、位运算。
    - **Registers**：极少量、超高速的专用/通用寄存器，用于暂存数据与地址。
    - **Cache**：小而快的就近存储，降低访存延迟（见下节）。
    - **MMU（Memory Management Unit）**：配合 OS 管理虚拟内存（有的架构可选/集成）。
    - **Firmware**：上电自检、初始化等固件程序。
- 集成形态：现代 CPU 常以单芯片集成于 **motherboard** 上，与内存与总线协同工作。

2.1.2 Cache（高速缓存）

- 特性：容量小（KB–MB 级），速度比 RAM 快 10–100 倍；通常紧贴 CPU。
- 作用：自动保留“最近/常用”数据副本；对 Cache 的修改由硬件协议回写到主存。
- 直觉：若数据访问**有局部性**（temporal/spatial locality），Cache 命中率高→性能好。

2.1.3 Bus（总线）

- 定义：连接 CPU、内存、外设的通信电路；**带宽/延迟**直接影响整机性能。

2.1.4 Main Memory（RAM）

- 特性：随机访问、容量中等、相对快速；**易失性**（断电即失）。
- 备注：用户常说的“16 GB 内存”即指 RAM。

2.1.5 External Storage（外部存储）

- 类型：HDD/SSD/U 盘等；容量大、但相对 RAM **非常慢**。
- 物理约束：旋转磁盘与甚至 SSD 都对访问顺序较敏感，顺序 I/O 往往优于随机 I/O。

2.1.6 Ports（外设端口）

- 含义：与外围设备交互的可寻址接口；具体控制由 **device driver**（设备驱动）实现。

> **Remark**
>
> 性能三分法：**compute-bound / memory-bound / I/O-bound**。在数据密集工作负载中，Cache 命中与内存/总线带宽往往比“CPU 峰值算力”更关键。

2.2 How Programs Run（程序如何运行）

> **Logic**
>
> 程序本质是机器码指令序列，CPU 通过寄存器与内存协作逐条执行。语言与编译/解释器是在此之上的抽象。

2.2.1 Machine code 与指令格式

- 指令基本形态：`opcode [operands]`，其中 **opcode** 是硬连线操作编号，**operands** 可能是寄存器名、内存地址或立即数。
- 执行流程：CPU 获得程序入口地址 → 取指 → 译码 → 执行 → 更新状态（寄存器/内存/标志位）。

> **Example**
>
> 以 Intel x86 的 Euclid GCD（最大公约数）为例：机器码难读，汇编（assembly）只是在机器码上添加最低限度的助记符与语法。

Asm

```
pushl %ebp
movl  %esp, %ebp
; ... 省略若干指令（比较、条件跳转、减法循环）...
leave
ret
```

近“金属”的编程有助理解寄存器、堆栈、跳转等底层机制，但日常工作更依赖高级语言。

2.2.2 控制流关键寄存器

- **PC / IP（Program Counter / Instruction Pointer）**：指向下一条待执行指令；遇到分支/跳转/调用/返回会被相应修改。
- **SP（Stack Pointer）**：指向栈区顶端；函数调用时用于暂存参数、返回地址、临时变量等。

2.2.3 Programs that run other programs（编译/解释）

- **Compiler（编译器）**：如 C/C++ 编译为本机机器码 → 可执行文件。
- **VM（Virtual Machine）**：如 **JVM**，将字节码视为“理想化机器”的机器码，由运行时程序解释/执行。
- **Interpreter（解释器）/Scripting**：如 `python`、`Rscript`、`ruby` 等进程直接读取并执行更高层语言的指令。

> ⚠ **Remark**
>
> 实际系统中常见**混合执行**：JIT、解释与原生扩展（如 C 扩展、Cython、Rcpp）并存，以兼顾开发效率与性能。

2.3 Files and Directories（文件与目录）

- **File**：持久化的数据对象，具备权限与元数据。
- **Directory（Folder）**：包含文件/子目录的特殊文件；文件系统呈树状层级。
- **Path**：唯一地址，例如 `/Users/spock/logic-puzzles/Kolinahr` 表示 `root` → `Users` → `spock` → `logic-puzzles` → 文件 `Kolinahr`。
- GUI（图形界面）以“图标/文件夹”隐喻呈现；CLI/程序中更常使用路径与系统调用直接操作。

> **Example**
>
> `find /Users/spock -name "Kolinahr"` 可在命令行按路径树搜索文件；在 GUI 中则逐层点击目录导航。

2.4 Operating System（操作系统的角色）

> **Logic**
>
> OS 作为“资源管理者 + 抽象提供者”，把复杂硬件细节隐藏在统一的编程接口之下。

- 资源抽象：
  - **Virtual Memory**：给每个进程提供近似“线性、独占”的地址空间；按需将页面在 RAM 与磁盘间换入换出（paging）。
  - **File Abstraction**：逻辑上连续，物理上可能分散，OS 负责映射与调度读写。
  - **Process / Thread**：调度执行、隔离与通信（IPC）。

- 能力暴露：通过 **system calls**（系统调用）请求资源或与外设交互（如打开文件、网络通信、进程控制）。

3 UNIX Command Line

Logic

本教程链接: https://computing.stat.berkeley.edu/tutorial-unix-basics/

Logic

本教程涵盖了类 UNIX（例如 Linux 或 MacOs）环境的基础知识, 并特别介绍了 UNIX command line interface 的使用

这是一种在计算机上执行操作和自动化任务的有效方式, 熟悉 command line operation 将使你能够以可重复的方式更快地完成任务

本教程的教材见 GitHub, 教学视频见 YouTube video 的 Episodes 1-3 (前 20 分钟)

3.1 Introduction

3.1.1 The Shell

1. Shell 是什么

- 在 UNIX 命令行上操作也被称为 "using the terminal" 和 "using the Shell"
- 在使用 UNIX-style operating system (例如 Linux 或 MacOS) 的 terminal window 时, 你与之交互的 UNIX program 即为 Shell
- Shell 位于你与操作系统之间, 是一个为你运行其他命令并显示结果的程序

2. Shell 的类型

- `bash` 非常常见, 并且在许多系统上是默认的
- 在较新的 MacOS 版本中, `zsh` 是默认的 Shell, 它是 bash 的扩展
- 其他 Shells: `sh`, `csh`, `tcsh`, `ksh`

3.1.2 Accessing a UNIX command line interface

访问 UNIX 命令行界面的方法如下:

- MacOS: 可以在 `Applications -> Utilities -> Terminal` 下找到 Terminal
- Windows:
  - If you have a sufficiently new version of Windows 10, you can use the Windows Subsystem for Linux, which will provide you with an Ubuntu shell running bash on your own machine.
  - If you have access to remote machines running Linux, you can login to them using programs such as MobaXTerm and Putty. Once logged in, you'll find yourself in a Terminal window on the remote machine.
- JupyterHub: 可以在 "New"下启动一个 Terminal session
- Cloud-based options: 可以尝试使用 Google Cloud Shell 等云服务

3.1.3 Getting started

进入终端后就会看到 "prompt", 表示 Shell 正在等待我们输入命令

有时 prompt 只有 `$`, 但它通常会包含 current user 和我们在 filesystem 中所在的 directory 的信息

- 如果看到的是 `>` 而不是 prompt, 这意味着 Shell 认为您还没有完成输入命令, 并且正在等待您输入更多信息
- 如果看到的是一个 newline 但没有任何其他内容, Shell 可能期望您输入一些文本以便它进行处理

Example

这是一个显示当前用户是 "scflocal", 在名为 "gandalf" 的机器上, 位于 user home directory (通常用 `~` 表示) 中的 "tutorial-unix-basics" (sub) directory 中的 prompt:

Shell

```
scflocal@gandalf :~/tutorial-unix-basics>
```

Remark: Tutorial code formatting

在教程的其余部分, 将不会显示命令前的 prompt, 输出 (如果有) 将跟随代码

Remark

如果不确定该做什么, 可以按 `Ctrl-c` 返回到 usual prompt

3.1.4 查询当前用户名

`whoami` 会打印出当前用户的 username:

Shell

```
whoami
```

3.2 Files and directories

3.2.1 查询目录

3.2.1.1 查询当前工作目录

任何时候, 你在 UNIX 命令行中都有一个 **working directory**, 它是你当前在 file system 中的位置

`pwd` (“print working directory”) 命令可以查询当前的工作目录:

Shell

```
pwd
```

```
/home/scflocal/tutorial-unix-basics
```

3.2.1.2 查询工作目录下的 files 和 subdirectories

`ls` 会列出工作目录中的文件 (和子目录)

Shell

```
ls
```

```
assets
_config.yml
example.text
example.txt
filename with spaces.txt
_freeze
```

```
_includes
index.qmd
index.rmarkdown
_layouts
mv_assets.sh
myfile
name of my file with spaces.txt
_quarto.yml
README.md
_sass
_site
```

3.2.2 访问目录

`cd` 可以使用 absolute path (绝对路径) 与 relative path (相对路径) 访问目录

3.2.2.1 访问 home directory

单独运行 `cd` 可以访问 home directory

Shell

```
cd
pwd
```

```
/home/scflocal
```

3.2.2.2 访问子目录

可以使用 `cd` 并加上子目录的名称, 这个子目录是相对于我们的工作目录找到的

Shell

```
cd tutorial-unix-basics
pwd
```

```
/home/scflocal/tutorial-unix-basics
```

3.2.2.3 访问嵌套子目录

可以使用 `cd` 并加上 nested subdirectories 的名称

Shell

```
cd
cd tutorial-unix-basics/assets
pwd
```

```
/home/scflocal/tutorial-unix-basics/assets
```

3.2.2.4 访问父目录

可以使用 `..` 访问任何目录的父目录

Shell

```
pwd
cd ..
```

```
3  pwd
```

```
1  /home/scflocal/tutorial-unix-basics/assets /home/scflocal/tutorial-unix-basics
```

可以使用相对路径让 `..` 的使用更复杂, 这里我们将先向上移动一个目录, 然后进入一个不同的子目录:

Shell

```
1  cd assets
2  cd ../_includes
3  pwd
```

```
1  /home/scflocal/tutorial-unix-basics/_includes
```

然后我们向上移动两级目录, 再进入另一个子目录

Shell

```
1  cd ../../Desktop  ## go up two directories and down
2  pwd
```

```
1  /home/scflocal/Desktop
```

⚠ **Remark**

以上所有示例都使用了相对路径, 根据你运行命令时的 current working directory 进行导航

3.2.2.5 使用绝对路径进行访问

使用绝对路径的方法是让路径以 `/` 开头, 如 `/home/scflocal`, 如果路径不以 `/` 开头, 则会被解释为相对于 current working directory 的相对路径

Shell

```
1  cd /home/scflocal/tutorial-unix-basics/assets
2  pwd
```

```
1  /home/scflocal/tutorial-unix-basics/assets
```

⚠ **Remark: Absolute paths are not robust**

在 script 中使用绝对路径通常是一个坏主意, 因为如果 script 在不同的机器上运行 (这些机器通常有不同的文件系统结构) 或以不同的用户身份运行 (这些用户通常有不同的主目录), script 可能无法正常工作

3.2.3 The filesystem

3.2.3.1 查询文件系统结构

文件系统本质上是一个 upside-down tree

可以使用 `tree` 查看树状的文件系统结构:

Shell

```
1  tree
```

- `.` 表示 current working directory (`tutorial-unix-basics`)
- 其中包含子目录 `asset` 、 `_includes` 、 `_layouts` 等
- 这些目录中又包含文件和进一步的子目录 (如 `assets`, 它包含名为 `css` 和 `fonts` 的子目录)

3.2.3.2 查询根目录 root directory

可以使用 `/` 表示整个文件系统的顶端, 也就是根目录

`/` 目录中包含许多子目录, 例如

- `/home` (包含了所有用户的 home directories (家目录), 属于某个用户的所有文件都存储在这里)
- `/bin` (包含 UNIX 程序，也称为 "二进制文件")
- `/tmp` 是一个存放只需要短暂使用且不需要保存的临时文件的地方, 当机器重启时, 这些文件会消失

Shell

```
ls /
```

3.2.3.3 查询家目录 home directory

如果有一个名为 `scflocal` 的用户, 则与该用户相关的一切信息都会存储在该用户的 **home directory** 中, 即 `/home/scflocal` (在不同系统中具体位置可能会有所不同)

- 可以使用 `~scflocal` 来表示 `scflocal` 的家目录, 即 `/home/scflocal`
- 如果你是 `scflocal` 用户, 可以通过快捷方式 `~` 来访问家目录

Shell

```
ls /home
```

```
scflocal
shiny
```

Shell

```
cd /home/scflocal
pwd
```

```
/home/scflocal
```

Shell

```
cd ~
pwd
```

```
/home/scflocal
```

Shell

```
cd ~scflocal
pwd
```

```
/home/scflocal
```

3.2.3.4 访问 tmp 目录

Shell

```shell
cd /tmp
ls
```

```
assets
assets.tgz
quarto-session71f9ea197eacbf29
RtmpMQcgrV
Temp-76c6318d-4f54-44d2-8bd9-d9a42eeeb7ce
test
```

3.2.3.5 返回最近所在的目录

可以使用 `cd -` 返回到最近所在的目录

Shell

```shell
cd -
pwd
```

```
/home/scflocal
```