Unix Overview

Bio724D: Fall 2023

2023-11-06

Unix history

Unix is an operating system, first developed at Bell Labs in the early 1970s. A few years after it's initial development, Bell Labs made the operating system and it's source code available to educational institutions. Because the source code was available, Unix became a popular platform for research and development in academia.

The history of Unix, its commercialization, various legal battles, etc are long and complicated. However, for our purposes it is sufficient to know that the most popular "descendant" of Unix used today is a free and open source operating system called Linux. Linux is built on the same principles as Uni, and taking advantage of many of the same tools and concepts which have been updated and improved on over the last 50 years.

Some key features that make Unix systems powerful

- Multi-user
- Supported on a wide array of hardware architectures
- Modular design
- Source code available, allowing fixes, improvements, derivations
- "Everything is a file" – documents, directories, system resources, physical devices, network interfaces, etc call all be accessed as if they were files. Shared tools and programmatic interfaces can be used across diverse resources.

Terminology: Kernel, Shell, Terminal

- Kernel – the kernel is the core or inner layer of an operating system. It controls all the tasks of the system such as managing memory and processes, communicating with attached hardware, providing useful abstraction through which other programs interact with the computer, etc
- Shell – a shell is program/environment that provides an interface between the kernel and the user. The shell interprets and translates user commands into underlying calls to the kernel.
- Terminal – A terminal is a text input/output environment for interacting with a computer. Originally, a terminal was a hardware device attached to a computer. Strictly speaking, these days we run "terminal emulators" – graphical programs that emulate terminals. We interact with the shell through the terminal.

Shells

- Bourne Again Shell (bash) – the default shell on our Linux VMs and probably the most popular terminal across different Unix operating systems. Derived from an older shell called the Bourne shell (sh).
- csh/tcsh – a shell with a C-like syntax; first developed by Bill joy at UC Berkeley as part of the Berkley System Distribution (BSD) Unix.
- Zsh – default shell on MacOS.
- … many others …

POSIX shell standard – an agreed upon set of standards for how shells should work. If you write shell scripts that follow the POSIX standard then in theory they should be easily portable across different shells that implement the standard.

The Unix Philosophy

This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface. – Doug McIlroy

The Unix Philosophy, Expanded

from McIlroy, Pinson, and Tague, 1978

1. Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new “features”.
2. Expect the output of every program to become the input to another, as yet unknown, program. Don’t clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don’t insist on interactive input.
3. Design and build software, even operating systems, to be tried early, ideally within weeks. Don’t hesitate to throw away the clumsy parts and rebuild them.
4. Use tools in preference to unskilled help to lighten a programming task, even if you have to detour to build the tools and expect to throw some of them out after you’ve finished using them.

Jesse and Greg’s material

… Introducing some more common commands and programs ….

Unix standard streams

Programs that follow Unix conventions and that execute in a shell have associated with them a set of what are called “standard streams”, which you can think of as channels for communicating with the outside world. The three standard streams are referred to as:

- standard input (`stdin`)
- standard output (`stdout`)
- standard error (`stderr`)

By default, `stdin` is usually associated with keyboard input and `stdout` and `stderr` are the terminal display, but these can be changed.

Redirection, output operators

- `>` – redirect the output operator. Sends the output of the command on the left to the file, device, or stream on the right. If the file already exists, it will be overwritten. If the file doesn't exist it will be created.
    - `echo "Hello, World!"` – by default, stdout is associated with the terminal so executing this command command prints the result of the echo command to terminal display.
    - `echo "Hello, World!" > hello.txt` – We're now redirecting stdout to a file called `hello.txt` (open this file to confirm that the contents are what is expected)
- `>>` – append output operator. Like redirect operator but appends output to the specified file rather than creating/overwriting.

```
echo "first line" >> lines.txt
echo "second line" >> lines.txt
```

Redirection, input operator

- < – redirect input operator
  - The `tr` (translate) doesn't have a built in mechanism for reading from a file, only from stdio. However we can redirect a file to stdio use the < operator. For example, here we read text from the file `hello.txt` (created above) and translate all upper case letters to their lower case equivalents:

```
tr '[:upper:]' '[:lower:]' < hello.txt
```

We can even chain together redirect operators like so:

```
tr '[:upper:]' '[:lower:]' < hello.txt > lower_hello.txt
```

Pipes

- `|` – pipe operator. The output (`stdout`) of the command on the left is used as the input (`stdin`) for the command on the right.
- The left and right commands run concurrently
- The data is buffered – only some of it is loaded into memory at a time

Examples:

- `ls *.txt | wc -l` – list all files ending in `.txt` and count how many they are using the `wc` utility
- `cat file.txt | fold -w 1` – read the input of a file and create output with one character per line
- `seq 5 15 | shuf` (but see `shuf -i` in man pages) – generate numbers between 5 and 15 and randomly shuffle them

Extended example: Exploring genome annotation

Use `wget` to download the genome annotation for the *Saccharomyces cerevesiae* (budding yeast) genome onto your VM (see course wiki for link).

GFF feature format

GFF format – “Generic Feature Format”, a plain text format for cataloguing recognized features in genomes. This is a tab-delimited file with 9 columns, the details of which are described in the GFF3 Specification.

We’re going to be most concerned with these columns:

- Column 1: “seqid” – The ID of the landmark used to establish the coordinate system for the current feature. e.g. “chrom1”
- Column 3: “type” – The type of the feature, e.g. “gene” or “exon”.
- Columns 4 & 5: “start” and “end”– The start and end coordinates of the feature are given in positive 1-based integer coordinates.
- Column 7: “strand” – The strand of the feature. + for positive strand (relative to the landmark), - for minus strand, and . for features that are not stranded.
- Column 9: “attributes” – A list of feature attributes in the format tag=value. Multiple tag=value pairs are separated by semicolons. See the GFF3 spec for more info.

Let’s build some pipelines to answer the following Q’s

- How many annotated features are there in the yeast genome?
- What is the set of feature types?
- How many gene features are there?
- Which chromosome has the most genes?