CSCI 330

UNIX Introduction

Jon Lehuta

Northern Illinois University

August 20, 2020

UNIX Introduction - Outline

Northern Illinois University

Introduction

- What is UNIX?
- The current state of UNIX/Linux as OS
- Basic commands
- How to get help

Northern Illinois University

What is UNIX?

UNIX is a family of operating systems that originated in the mid-20th century. It includes Linux, BSD, MacOS, and many others.

Northern Illinois University

What Can UNIX Do for You?

A UNIX operating system is a framework that can allow you to do many things, for example:

User Tools

- Text processing (`vi`, `sed`, `awk`)
- Productivity applications (web, mail, chat)

Programming Tools

- compilers (C, C++, Java)
- source code control systems (`git`, `svn`, `cvs`)

Internet Servers

- mail server
- web server
- database server

Northern Illinois University

What is an Operating System

- An *operating system* is the software that is responsible for managing (allocation of, deallocation of, and access to) system resources in an efficient and secure manner.

What type of resources?

The operating system is responsible for managing:

- Which program gets to use the CPU(s) at a given time
- To which parts of memory each program has access
- Where data can be stored
- Which programs can use the network
- ... and more!

Northern Illinois University

History of UNIX

First UNIX invented by Ken Thompson at Bell Labs in 1969

- first version written in assembly language
- single user system, no network capability

Thompson, Dennis Ritchie, Brian Kernighan

- rewrote UNIX in C (also invented C)

UNIX evolution:

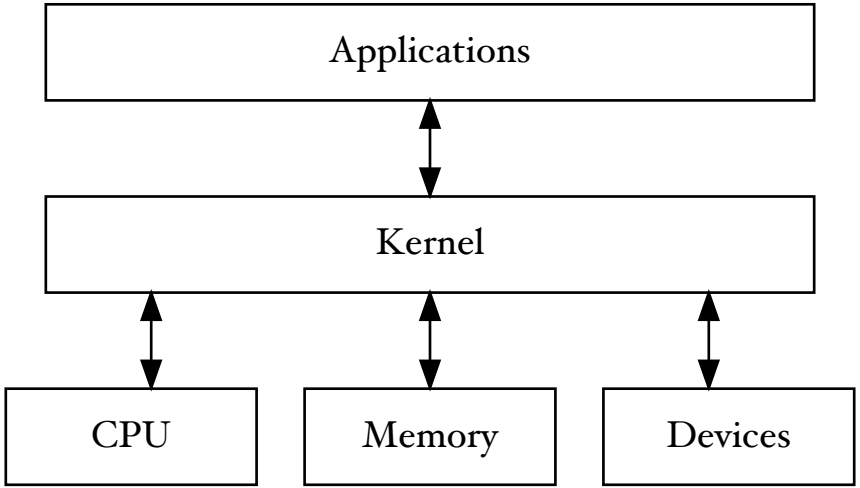
- Bell Labs, USL, Novell, SCO
- BSD, FreeBSD, Mach, OS X
- AIX, Ultrix, Irix, Solaris
- Linux: Linus Torvalds

Newest:

- Linux on portables: Android
- Darwin (BSD) on iPhone/iPad/Mac

Northern Illinois University

Components of UNIX



The basic organization of a UNIX operating system.

Northern Illinois University

Types of UNIX

- GNU/Linux – based on kernel created by Linus Torvalds, and using the GNU project's implementations of system commands.
  - We will be using Linux for this course.
- BSD – Berkeley-Style Daemon - based on source code released by Berkeley. MacOS X uses a lot from this.
- Commercial – AT&T, Solaris, etc.

Northern Illinois University

Linux Distributions

A Linux *distribution* includes the *kernel* and a collection of useful software.

Redhat-style – `rpm` packages and `yum` package manager

- Redhat Enterprise Edition
- Fedora
- CentOS

Debian-style – `deb` packages, `apt` package manager

- Debian
- Ubuntu
- Mint
- KDE Core

Others

- Gentoo
- Slackware
- OpenSUSE

Northern Illinois University

Different Types of Operating Systems

Single-user, single-process operating systems:

- allow only one user at a time on the computer system
- user can execute/run only one process at a time

Examples: DOS, MacOS 9

Single-user, multi-process operating systems:

- allow a single user to use the computer system
- user can run multiple processes at the same time

Example: OS/2

Multi-user, multi-process operating systems:

- allow multiple users to use the computer system simultaneously
- Each user can run multiple processes at the same time

Examples: UNIX, Windows NT (2000, XP, Vista, 7, 10)

Northern Illinois University

Multiprocessing operating system

UNIX operating systems do something called *multiprocessing*. This means that many programs can be running (seemingly) at the same time.

Each of these programs runs inside its own *process* which contains:

- Information on the current state of the program (program counter, register values, etc.)
- Memory that is private to the program that is running
- Information on which files the program has open at the moment

The memory for a process is not available from another process without asking for it through the operating system. We will talk about interprocess communication (IPC) later in the course.

Multiuser operating system

UNIX operating systems are also *multiuser*. Different users can be using a UNIX system concurrently (at the same time), and their access to various resources can be configured on a per-user basis.

Northern Illinois University

The Shell

- There are various ways of interacting with the operating system. We can do this by writing programs, or we can use programs that are already written for us. Generally, we will use something called a *shell* to run programs to accomplish our tasks.
- Doing things with the shell is usually called working with the *command line*
- The shell that we will be using for this class is called `bash`.

Features of a typical shell:

- Interpret and execute commands
- Command history and editing
- Job control
- Scripting

Basic Shell Usage

- The shell is a program that is run automatically for you by the operating system when you log in.
- It prints a prompt for you, then allows you to type text, waiting for you to hit enter.
- Once you hit enter, it tries to evaluate what you have typed and interpret it as instructions to perform.
- If what you typed is valid, the shell will attempt to have the operating system perform the task for you.

Northern Illinois University

Shell Terminology

The shell is attached to a *terminal*. The terminals we use nowadays are virtual, but they used to be physical machines.

The terminal's job is to be a screen that does these things (normally):

1. Read things you type and send them to whatever program is currently in the foreground through a stream called standard input. (C++ programs read standard input with `cin`)
2. Display things that come through the stream called *standard output*. (In C++, `cout` prints to standard output.)
3. Display things that come through the stream called *standard error*. (In C++, `cerr` prints to standard output.)

UNIX Shells

`sh`-style

- Bourne shell: Steve Bourne, 1978
- Almquist shell (ash): BSD `sh` replacement
- Bourne-Again shell (`bash`): GNU/Linux

`csh`

- C shell, Bill Joy, BSD, 1978
- Tenex C shell (tcsh): GNU/Linux

Others: Korn shell (`ksh`), Zshell (`zsh`), `fish`

Simple Shell Commands

We will learn much more about what we can type on the shell later, but the simplest thing we can do at the shell is run commands. Below is a table with some simple ones that can do things without needing more information.

Command	**Action**
`ls`	list the files in the current directory
`pwd`	print the *path* of the *current working directory*
`clear`	clears whatever was being displayed by the terminal
`date`	prints the current date and time
`passwd`	change password of current user
`exit`	exits the current shell instance, logging out if none left

Notice that all of these commands are single words with no spaces. This will be important.

Command Line Arguments

Sometimes, when a program launches, it needs more information to function. It could read these things from files, or prompt the user for them, but often the more convenient way is to supply the extra information as *command line arguments*.

The commands from the previous slide could each be run on their own. Some of them can change their behavior if you supply more information.

Command	**Action**
`ls -l`	list more information on files in the current directory
`ls -a`	show *all* of the files in this directory, even if hidden
`ls file`	shows the file named `file` if it exists, or an error
`date +%F`	prints the current date in YYYY-MM-DD format
`passwd x`	prompts for information to change password of user `x`

Command Line Arguments

As far as the shell is concerned, the *command line arguments* are just strings, separated by spaces, that come after the name of the command.

When the program gets them, they come in as parameters to the `main` function. (In C++, anyway.)

```
int main(int argc, char *argv[]) {
  // argc = argument count, number of command line arguments + 1
  // argv = argument vector = array of null-terminated strings
  // argv[0] = name of command, as typed at the shell
  // argv[1] = first command line argument as char *
  // argv[2] = second command line argument as char *
  // argv[i] = ith command line argument, if i < argc
}
```

What a program does with them is up to the person who designs it, but that is how it will access the information.

Northern Illinois University

Command Line Arguments Example

```
echo Thanks for all the fish.
```

Typing the above at the shell, `echo` runs, and in its `main`, it will have an `argc` of 6.

i	argv[i]
0	"echo"
1	"Thanks"
2	"for"
3	"all"
4	"the"
5	"fish."

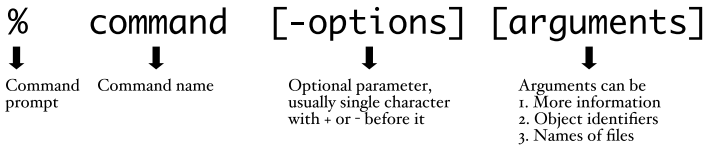
The `echo` command prints its command line arguments to *standard output*, like `cout`, but for the shell. It would print

```
Thanks for all the fish.
```

Northern Illinois University

Command Line Structure

We now know how to access command line arguments inside a C/C++ program. Programs that are already written tend to use command line arguments like the following:



Basic structure of a command line expression.

- UNIX is case sensitive
- Must be a space between command, options and arguments
- No space between the plus or minus sign and option letter
- Fields enclosed in `[]` above are *optional*, the `[` and `]` aren't typed.
- Many programs use a library function called `getopt` to detect options.
- Don't type the `%`, it is just indicating that you're typing it at the shell prompt.

Northern Illinois University

Command Line Example

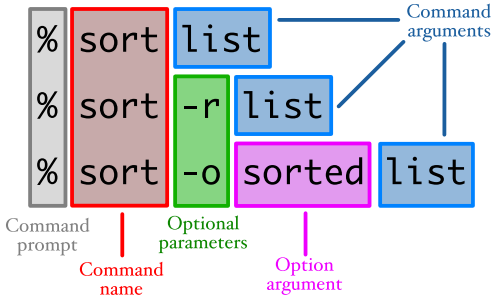


Diagram of several command line expressions split into their parts.

- Here, the `sorted` after `-o` is an *option argument*. This is because, the way that the `sort` program is designed, the `-o` option requires more information, and takes it from the next argument supplied.

Northern Illinois University

Some Basic Commands

`passwd`	change password
`pwd`	print absolute path of current working directory
`ls`	list files
`more`	show content of file, page by page
`logout`	logout from system
`date`	display date and time
`who`	display who is on the system
`clear`	clear terminal screen
`script`	make record of a terminal session
`uname`	print current OS detail (version etc.)
`man`	find and display system manual pages
`exit`	exits the current shell instance, logging off if none left

RTFM: The man Command

man - show pages from system manual

Syntax: `man [options] [-S section] command-name`

```
% man date
% man crontab
% man -S 5 crontab
```

Section	Purpose
1	User commands
2	System calls
3	C library functions
4	Special system files
5	File formats
6	Games
7	Miscellaneous features
8	System administration

- Note: Some commands are aliases
- Note: Some commands are builtin to the shell and can be found in the shell's manpage

Northern Illinois University

The UNIX File System

Hierarchical organization of files

- contains directories and files and devices
- **EVERYTHING** is a file
- always single tree

Basic commands to list and manipulate files

- independent of physical file system organization

Typical UNIX file system types

- ext4 (formerly ext2, ext3)
- reiserfs
- also: vfat, ntfs

Northern Illinois University

Directory terminology

Root Directory: /

- top-most directory in any UNIX file structure

Home Directory: ~

- directory owned by a user
- default location when user logs in

Current Directory: .

- default location for working with files

Parent Directory: ..

- directory immediately above the current directory

Path

Path: list of names separated by "/", that locates/identifies a file on the file system

Absolute Path

- Traces a path from root to a file or a directory
- Always begins with the root (/) directory

Example: `/home/student/Desktop/assign1.txt`

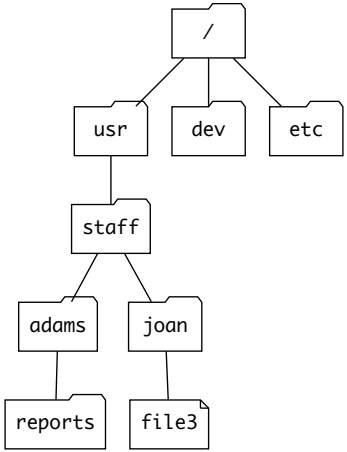
Relative Path

- Traces a path from the current directory
- No initial forward slash (/)
- dot (.) refers to current directory
- two dots (..) refers to one level up in directory hierarchy

Example: `Desktop/assign1.txt`

Northern Illinois University

Absolute Path Example



Example file system.

The absolute path to `file3` is `/usr/staff/joan/file3`

Northern Illinois University

Some path-related commands

`pwd`	to show path of current working directory
`cd`	to change the current working directory
`mkdir`	to create a new directory
`rmdir`	remove a directory (when it's empty)

- use `rm -r` to remove non-empty directories

Northern Illinois University

List directory content

The most frequently used file system command: `ls`

Displays names of files present on the file system

Syntax: `ls [options] [path]`

- displays the files/directories specified as the `path` parameter
- if `path` is not supplied, lists files in current directory
- displays an error if what `path` points to is not present

Northern Illinois University

`ls` options

Common options:

`-a`	show all files
`-l`	show long version of listing
`-t`	show files sorted by time stamp
`-S`	show files sorted by file size
`-r`	show files in reverse sorted order

Northern Illinois University

Long List Option

List contents of the current directory in long format

```
% ls -al
drwxr-xr-x 13 user group 1024 Apr 26 15:49 .
drwxr-xr-x 15 root root   512 Apr 24 15:18 ..
-rwx------  1 user group 1120 Apr 12 13:11 .config
-rw-r--r--  1 user group  141 Mar 14 13:42 .logout
-rwx------  1 user group  436 Apr 12 11:59 .profile
drwx------  7 user group  512 May 17 14:11 330
drwx------  2 user group  512 Mar 31 10:16 Data
-rw-r--r--  1 user group   80 Feb 27 12:23 quiz.txt
```

- `.` is current dir
- `..` is parent dir
- directories have a `d` in first column
- names beginning with a dot (`.`) are hidden
- plain files have a `-` in the first column

File System Commands, `cp`

Syntax: `cp source target`

`source` is one or more paths for items to copy

`target` is where to put the copy/copies:

- if only copying a single file and `target` does not exist, it is created and becomes a copy of the original
- if only copying a single file and `target` exists,
  - if `target` is a normal file, it is overwritten with the data in `source`
  - if `target` is a directory, the `source` file will be copied into that directory with the same name
- if copying multiple files, `target` *must* be a directory

Commonly used options:

`-i`	if `target` exists, the command cp prompts for confirmation before overwriting
`-R`	recursively copy entire directories
`-p`	preserve access times and permission modes

Moving/Renaming files/directories, `mv`

There is no special command for renaming, but you can accomplish the task with the move command, `mv`.

Example: Rename file `unix` to `csci330`.

```
% mv unix csci330
```

Caveat:

What happens if `csci330` exists and is a directory?

Deleting files, `rm`

Syntax: `rm [options] path-list`

Where `path-list` is a list of paths to files that are to be removed.

Commonly used options:

`-f`	force remove regardless of permissions
`-i`	prompt for confirmation before removing
`-r`	"recursive" removes everything under the indicated directory as well

Without the `-r` flag, `rm` will emit an error when used on directories.

Example: Remove the file, `old-assign`

```
% rm unix/assign/old-assign
```

Northern Illinois University

Deleting empty directories, `rmdir`

Syntax: `rmdir [options] emptydir`

If `emptydir` is not a directory, or if `emptydir` has other files inside it, `rmdir` will fail.

Deleting non-empty directories, `rm -r`

Syntax: `rm -r directory`

This will delete `directory` and, if it is a directory, everything inside of it.

Northern Illinois University

Linking Files with `ln`

A *link* is a tool that allows a file (or directory) to be referenced by another name.

It is a special type of file that, when read or changed, affects another file on the system instead.

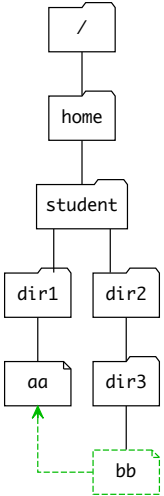
Two types:

- *Hard link* - made by `ln` without `-s` option present
- *Symbolic link* (sometimes incorrectly called a "soft link") - supply the `-s` option

Syntax: `ln [-sf] target linkname`

Northern Illinois University

Link illustration



In the above, `bb` in `dir3` is a link to file `aa` in `dir1`

File System Layout

Files on UNIX file system consist of:

- data blocks
- identified by block id
- file meta information: *inode*, containing:
  - which blocks make up file
  - permissions, etc.
- stored in *inode table*
  - index into table is *inode number*

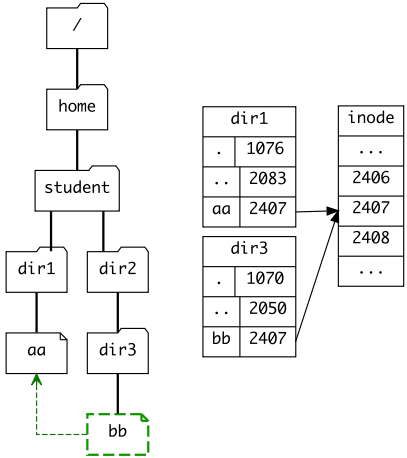
A directory is table of:

- *file name* → *inode number*

Northern Illinois University

Hard Link example: `ln`

```
% ln /home/student/dir1/aa /home/student/dir2/dir3/bb
```



Notice that a hard link to a file will share the inode from the original file.

Symbolic Link example: `ln -s`

Symbolic links do not share an inode with their target. Instead, the path to the target is stored. This allows us to make links across physical devices with different inode tables, but changes the behavior of the links as a consequence.

To see where a symbolic link points, you can use `ls -l`.

```
% ls -l bb

lrwxrwxrwx 1 user group 22 Nov 17  2018 bb ->
/home/student/dir1/aa
```

Notice the `l` in the first column, and the `->` at the end, indicating the target.

Link type comparison

Hard Link	Symbolic Link
► Target file must exist upon link creation, in order to know which inode. ► Original file will continue to exist as long as any hard link to it exists. ► Cannot link to a file located on a different physical device. ► Cannot circularly link to another hard link.	► Can be created even before the target file exists. ► Cannot access the target if it is missing or if the user doesn't have permission for the file. ► Can link across physical file systems. ► Can be circularly linked to another symbolically-linked file.

Finding Files

Syntax: `find path-list expression(s)`

`find` recursively descends through directories in `path-list` and applies the supplied expression for every file

Get details on available expressions by typing `man find`.

Northern Illinois University

More Commands

- `head` - show some of the lines at the beginning of a file
- `tail` - show some of the lines at the end of a file
- `wc` - short for *word count*, display number of words, characters, lines present in input data

Even More Commands

- `touch` - update the modification date on a file without changing its contents, or create a file if it doesn't exist
- `cat` - show the contents of specified file(s), in the order specified
- `more`, `less` or `pg` - show data one page at a time

Comparing Files: `diff`

Compare two files line by line, showing the differences.

Syntax: `diff [options] file-1 file-2`

- If `file-1` and `file-2` have the same contents, no output is produced
- If `file-1` and `file-2`'s contents are not the same, `diff` reports a series of commands that can be used to convert the first file to the second file (via the `patch` command)

Northern Illinois University

Compress File Contents

utilities to compress and uncompress files common on Linux:

- `gzip`, `gunzip`, `zcat`
- file extension: `.gz`

Example:

```
% gzip assign1.txt
% zcat assign1.txt.gz
% gunzip assign1.txt.gz
```

Also:

- Smaller compressed file than `gzip`: `bzip2` or `lzma`
- Windows compatible: `zip`/`unzip`, `rar`/`unrar`

Sorting Files

Syntax: `sort [options] file-name`

Commonly used options:

`-r`	sort in reverse order
`-n`	numeric sort
`-t`	field delimiter (default: blank)
`-k` x	sort based on value in field/column x
`-f`	ignore case

User's Disk Quota

quota is upper limit of

- amount disk space
- number of files

for each user account

The command: `quota -v`

- displays the user's disk usage and limits

2 kinds of limits:

- Soft limit: ex. 100MB
  - May be exceeded temporarily
  - System will nag
- Hard limit: ex. 120MB – Cannot be exceeded

Extra Info Online

`http://www.unixtools.com`

`http://www.ugu.com`

`http://www.tldp.org`

`http://linux.die.net`