Introduction to UNIX

Benjamin Brewster

UNIX - 1965

- AT&T Bell Laboratories, MIT and General Electric develop a new time-sharing OS called Multics (MULTiplexed Information and Computer Services), which was the first OS to provide a hierarchical file system.

MIT development led by Fernando J. Corbató

Image by Jason Dorfman, CC BY-SA 3.0

UNIX - 1969

Image by user Toresbe, CC SA 1.0, via Wikimedia Commons

PDP-7 under restoration

- Bell Labs management pulls out of the Multics project due to delays, but Bell Labs researchers continue to experiment on their own.
- Bell Labs’ Ken Thompson, Dennis Ritchie, Rudd Canaday, and Doug McIlroy create UNIX on a PDP-7.
- While Multics was a large project; UNIX was just a few researchers (carrying on UNiplexed Information and Computer Services).

UNIX - 1970s

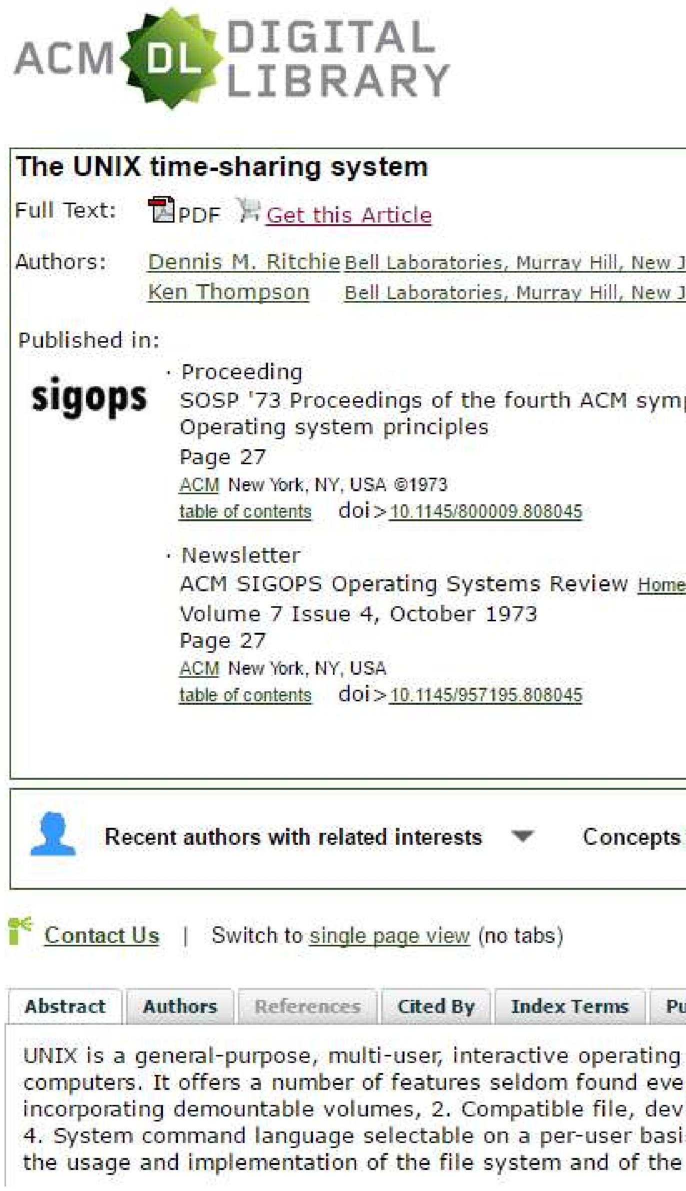
- January 1, 1970 becomes "time zero" for UNIX.
- Text processing abilities added, allowing UNIX to start being used for actual work by another department at Bell Labs.
- UNIX now runs on a PDP-11/20.

Image by Peter Hamer, CC BY-SA 2.0

Ken Thompson (sitting) & Dennis Ritchie at PDP-11

UNIX - 1970s

- First manual produced as online “man pages” On November 3, 1971.
- UNIX re-written in C (created by Dennis Ritchie) in 1972, spurring further development in the C language itself.
- Presented to the outside world at the 1973 Symposium on Operating Systems Principles, but could not be turned into a product because of anti-trust regulations.



The UNIX time-sharing system

Full Text: PDF Get this Article

Authors: Dennis M. Ritchie Bell Laboratories, Murray Hill, New J
Ken Thompson Bell Laboratories, Murray Hill, New J

Published in:

sigops

· Proceeding
SOSP '73 Proceedings of the fourth ACM sym
Operating system principles
Page 27
ACM New York, NY, USA ©1973
table of contents doi>10.1145/800009.808045

· Newsletter
ACM SIGOPS Operating Systems Review Home
Volume 7 Issue 4, October 1973
Page 27
ACM New York, NY, USA
table of contents doi>10.1145/957195.808045

ACM DIGITAL LIBRARY
Recent authors with related interests
Concepts
Contact Us | Switch to single page view (no tabs)
Abstract
Authors
References
Cited By
Index Terms
Pu

UNIX is a general-purpose, multi-user, interactive operating
computers. It offers a number of features seldom found eve
incorporating demountable volumes, 2. Compatible file, dev
4. System command language selectable on a per-user basi
the usage and implementation of the file system and of the

http://dl.acm.org/citation.cfm?doid=800009.808045

Linux

- Initially developed by Linus Torvalds to be a UNIX-like system, but not beholden to expensive license agreements with AT&T.
- Open source, protected by the GNU Public License (GPL)
- Supports the POSIX UNIX specification
  - Code written for another POSIX-based UNIX (ie Solaris, HPUX, AIX, etc) shouldn't need many changes to run on Linux.
  - Linux knowledge will apply to most UNIX systems
- Stable, robust, free
- First version released March 14, 1994.

Linus Torvalds speaking at LinuxCon Europe 2014

You Can Run Linux at Home

- There are many Linux distributions
  - Ubuntu, openSUSE, Red Hat, Debian, Arch Linux, Chrome OS, Mint, etc.
  - All can be downloaded; some can be purchased
  - EECS at OSU has several Linux machines you can connect to

- Distros are created with different purposes and features in mind
  - Ubuntu installs everything easily
  - Arch Linux is the opposite, requiring everything to be installed manually
  - Tails is built to be private from the ground up

What Is The Shell?

- The user interface to the operating system - a text-based, command line interpreter

```
$ □
```

- Provides access to all UNIX user-level programs
  - Start programs
  - Manage processes (job control)
  - Connect together processes (pipes)
  - Manage I/O to and from processes (redirection)
  - Kill programs

Shell Prompts

- Traditional prompt and cursor – your commands go *after* the prompt:

```
$ □
```

- Can be customized:

```
[1459][brewsteb@flip1:~]□
brewsteb@flip1@12:42:17$ □
~/CS344/prog1$ □
```

- A great place to create prompt code: http://ezprompt.net/
- For example:

```
export PS1="\u@\h@\T\\$ "
```

- Place custom prompts with this line in ~/.bashrc

Shell Examples

- Type in a command after the prompt ($), and bash executes it with the output:

List all files

```
$ ls
CS344   CS419   CS372
```

List all files, including hidden files

```
$ ls -a
.    ..    .bashrc    CS344    CS419    CS372
```

Different UNIX Shells

- Basic Shells
  - Bourne Shell (/bin/sh)
  - C-Shell (/bin/csh)

- Enhanced Shells
  - BASH “Bourne-again shell” (/usr/local/bin/bash)
  - TCSH enhanced C-Shell (/usr/local/bin/tcsh)
  - Korn Shell (/bin/ksh)

bash *was written by Brian Fox as an upgrade to Stephen Bourne’s shell* “sh”, *which itself was a replacement for Ken Thompson’s* “sh” *shell, which was based on the* “sh” *shell from Multics.*

Most Common UNIX Commands

- Directory/file management
  - cd, pwd, ls, mkdir, rmdir, mv, cp, rm, ln, chmod
- File viewing and selecting
  - cat, more/less, head, tail, grep, cut
- Editors
  - vi, (x)emacs, pico, textedit
- Other useful commands
  - script, find, telnet, ssh, and many more!

Common UNIX Commands - Directories & Files

- **pwd** - Print working directory. The “where am I” command; similar functionality can be integrated into the shell prompt.
- **cd** - Change directory. Moves your current working directory to a different one; accepts relative and absolute arguments.

```
$ cd ~/CS344/prog1
$ pwd
/nfs/stak/faculty/b/brewsteb/CS344/prog1
$ cd ..
$ pwd
/nfs/stak/faculty/b/brewsteb/CS344
$ cd ~
$ pwd
/nfs/stak/faculty/b/brewsteb
```

~ is a shortcut to your home directory

/ is the root of the entire directory structure

Common UNIX Commands - Directories & Files

- **ls** - Displays the files in a given directory. Accepts relative and absolute arguments.

```
$ ls
CS344    CS464    pidtest
$ ls -pla
drwx--x--x. 1 brewsteb upg57541   896 Jul 14 10:35 ./
drwxr-xr-x. 1 root     root      1100 Jun 22 21:38 ../
-rw-r--r--. 1 brewsteb upg57541  2431 Aug 18  2015 .bashrc
drwx------. 1 brewsteb upg57541  4153 Aug 13  2015 CS344/
drwx------. 1 brewsteb upg57541  9472 Jul 12 18:49 CS464/
-rwxrwx---. 1 brewsteb upg57541  6561 Mar  2 09:43 pidtest
$ ls -pla --color=auto
drwx--x--x. 1 brewsteb upg57541   896 Jul 14 10:35 ./
drwxr-xr-x. 1 root     root      1100 Jun 22 21:38 ../
-rw-r--r--. 1 brewsteb upg57541  2431 Aug 18  2015 .bashrc
drwx------. 1 brewsteb upg57541  4153 Aug 13  2015 CS344/
drwx------. 1 brewsteb upg57541  9472 Jul 12 18:49 CS464/
-rwxrwx---. 1 brewsteb upg57541  6561 Mar  2 09:43 pidtest
```

Common UNIX Commands - Directories & Files

- **alias** - Create a shortcut for running a program.

```
$ alias l="ls -pla --color=auto"
$ l
drwx--x--x. 1 brewsteb upg57541   896 Jul 14 10:35 ./
drwxr-xr-x. 1 root     root      1100 Jun 22 21:38 ../
-rw-r--r--. 1 brewsteb upg57541  2431 Aug 18  2015 .bashrc
drwx------. 1 brewsteb upg57541  4153 Aug 13  2015 CS344/
drwx------. 1 brewsteb upg57541  9472 Jul 12 18:49 CS464/
-rwxrwx---. 1 brewsteb upg57541  6561 Mar  2 09:43 pidtest
```

Common UNIX Commands - Directories & Files

- **`mkdir`** - Create directories
- **`rmdir`** - Delete directories
- **`rm`** - Delete files (and directories if used recursively)
- **`mv`** - Move or rename files and directories
- **`cp`** - Copy files and directories

- Example on next page…

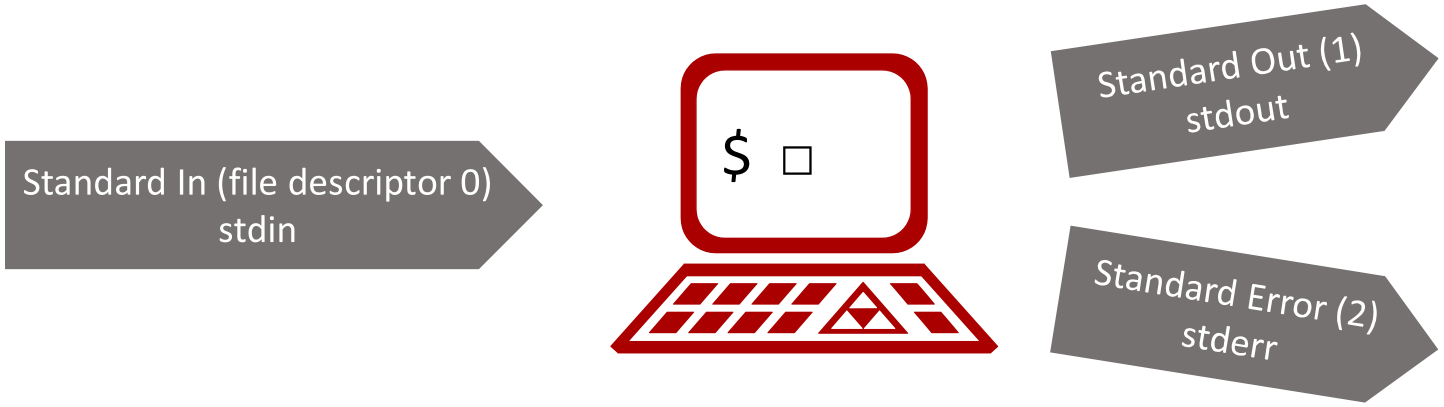
Common UNIX Commands - Directories & Files

```
$ mkdir tempdir
$ touch myfile
$ mv myfile tempdir
$ cd tempdir
$ alias l="ls -pla --color=auto"
$ l
drwxrwx---. 1 brewsteb upg57541  80 Jul 14 11:29 ./
drwxrwx---. 1 brewsteb upg57541 104 Jul 14 11:29 ../
-rw-rw----. 1 brewsteb upg57541   0 Jul 14 11:29 myfile
$ cp myfile myfile_backup
$ rm myfile
rm: remove regular empty file `myfile'? y
$ ls
myfile_backup
$ mv myfile_backup myfile
$ ls
myfile
$ cp myfile ../newfile
$ cd ..
$ ls
newfile tempdir
$ rmdir tempdir
rmdir: failed to remove `tempdir': Directory not empty
$ rm -rf tempdir
```

The `alias` command can be placed into ~/.bashrc where it will be run each time you log in to this computer. Handy!

Aside: Standard files and the Terminal

- The shell automatically opens the terminal for reading-from, with one file, and for writing-to, with two files:



- High level: If you don’t specify otherwise, program input and output goes to and from your shell prompt by default on every line
- We view and manipulate this input and output with the new few programs/commands

Common UNIX Commands - Getting Data from Files

- **echo** - send character data to standard out

```
$ echo text
text
$ echo test text
test text
$ echo -e "test text\nnext line"
test text
next line
$
```

`-e` means interpret special characters;
`\n` is a newline

- `echo` sends its data to stdout, which has not been explicitly changed away from the terminal.
- The terminal dutifully displays data sent to it on your screen (or on another network-attached screen, but that’s beyond our scope).

Common UNIX Commands - Getting Data from Files

- **`cat`** - concatenate character data stored in a file with character data from other files.
- Used primarily to dump data to the terminal.

```
$ cat file1
file1contents
$ cat file1 file2
file1contents
file2contents
```

- `cat` sends its data to stdout, which wasn't explicitly changed from the terminal, so out to our screen it comes.

Redirecting stdout

- The > operator tells stdout to open a different file for writing to (affects entire line).

```
$ ls
$ echo -e "cookie\nbeefcake\napple" > foodlist
$ ls
foodlist
$ cat foodlist
cookie
beefcake
apple
```

stdout redirected to open a file named foodlist for writing instead of the terminal

No redirection here, so cat's stdout has the terminal open for writing

Common UNIX Commands - Getting Data from Files

- **`sort`** - Takes data from stdin OR a file and sends the data, alphabetically sorted by line, to stdout.

```
$ echo -e "cookie\nbeefcake\napple" > foodlist
$ cat foodlist
cookie
beefcake
apple
$ sort foodlist
apple
beefcake
cookie
```

Redirecting stdin

- The < operator tells stdin to open a different file for reading from (affects entire line).

```
$ sort foodlist
apple
beefcake
cookie
```

Here, `sort` opens up the file with filename given in the first argument, reads the data in, sorts it, and sends it to stdout

```
$ sort < foodlist
apple
beefcake
cookie
```

Here, bash opens up the named file, and sends the data on as input to the program on the left.

`sort`, meanwhile, has no first argument, so it reads data from stdin, and does not know where the data comes from! It gets sorted and sent to stdout.

stdin with no Redirection

```
$ cat > list
cookie
beefcake
apple
$ cat list
cookie
beefcake
apple
```

`cat`, having no argument, attempts to read from stdin, however stdin still has the terminal open for input. Thus, the input comes from the keyboard, line by line: you type each element in, hitting return each time, and stop by typing `^d`

Afterwards, bash takes the data from the program on the left, opens the named file, and writes the data into it.

Redirecting both stdin and stdout

```
$ echo -e "cookie\nbeefcake\napple" > foodlist
$ sort < foodlist
apple
beefcake
cookie
$ sort < foodlist > sortedList
$ cat sortedList
apple
beefcake
cookie
```

1. `sort` has no file given to it as an argument, so it tries to read from stdin
2. stdin for this line gets redirected from the file foodlist
3. The file contents get fed into `sort`, which sorts them.
4. stdout is redirected to the file sortedList,
5. `sort`'s output is written to stdout, which goes to sortedList

```
$ sort >sortedList <foodlist
$ cat sortedList
apple
beefcake
cookie
```

The order of the stdin and stdout redirections doesn't matter. You can leave out the space, too, if that makes thing more clear

Shell Filename Expansion

- Certain metacharacters are expanded and replaced with all files with matching names
- * - matches anything
- ? – matches any one character

- Note that this isn't a regular expressions encoding - that's performed with different programs, most notably `grep`

Shell Filename Expansion Examples

- List all files in the current directory whose names start with “CS344”:

```
$ ls CS344*
```

- List all files in the current directory that have “CS344” somewhere in their name:

```
$ ls *CS344*
```

- In all subdirectories that contain “CS344“ followed by any character, list all files:

```
$ mkdir CS344; touch CS344/fileInCS344
$ mkdir CS3440; touch CS3440/fileInCS3440
$ mkdir CS34401; touch CS34401/fileInCS34401
$ ls */*CS344?
CS3440/fileInCS3440
```

Pipes

- Provide a way to communicate between commands without using temporary files to hold the data

```
$ echo -e "cookie\nbeefcake\napple" | sort
apple
beefcake
cookie
$ echo -e "cookie\nbeefcake\napple" > foodlist
$ cat foodlist | sort
apple
beefcake
cookie
$ cat foodlist | sort > sortedList
$ cat sortedList
apple
beefcake
cookie
```

A pipe takes the stdout of one command and connects it to the stdin of the next

`more`, append, and pipes

- **`more`** - Takes character data and displays only one screen-full at a time; navigate with up, down, & spacebar; quit with q.

- >> operator *appends* character data to the end of a file, as opposed to *replacing* the contents of an existing file as > does.

- Shell demo script:

```
$ echo -e "1\n2\n3\n4\n5" > rowfile
$ echo -e "6\n7\n8\n9\n10\n11\n12\n13\n14\n15" >> rowfile
$ echo -e "16\n17\n18\n19\n20\n21\n22\n23\n24\n25" >> rowfile
$ cat rowfile | sort -nr >> rowfile
$ cat rowfile
$ cat rowfile | more
```

Live Shell Demo Script

```
$ bash
$ echo -e "cookie\nbeefcake\napple" > foodlist
$ echo -e "dogfish\neel" > fishlist
$ ls; ls
$ cat foodlist fishlist
$ sort < cat foodlist fishlist
-bash: cat: No such file or directory
$ cat foodlist fishlist > biglist
$ cat biglist
$ sort < biglist > sortedBiglist
$ cat sortedBiglist
$ sort -r < biglist > reverseSortedBiglist
$ cat reverseSortedBiglist
$ cat *list *list | sort | more
```

By way of example, this is a bad command: there is no file called "cat", and the rest is ignored!