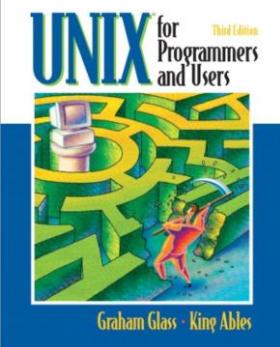


UNIX



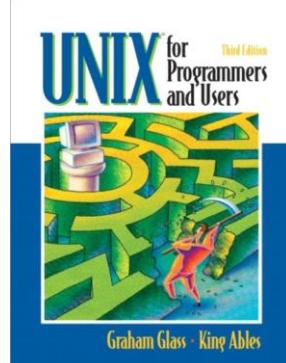
YORK **U**
UNIVERSITÉ
UNIVERSITY

1

Contents

- Overview of UNIX
 - Structures
 - File systems
 - Pathname
 - Security
 - Process:
 - Exit code
 - Pipes
- Utilities/commands
 - Basic
 - Advanced
- Shell and shell scripting language

today

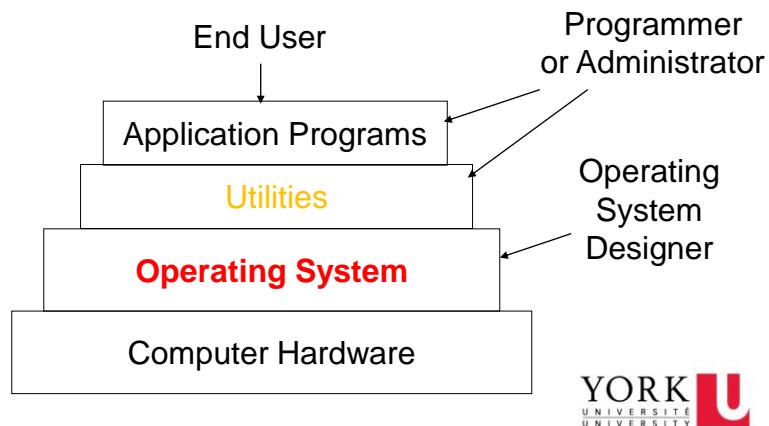


YORK **U**
UNIVERSITÉ
UNIVERSITY

2

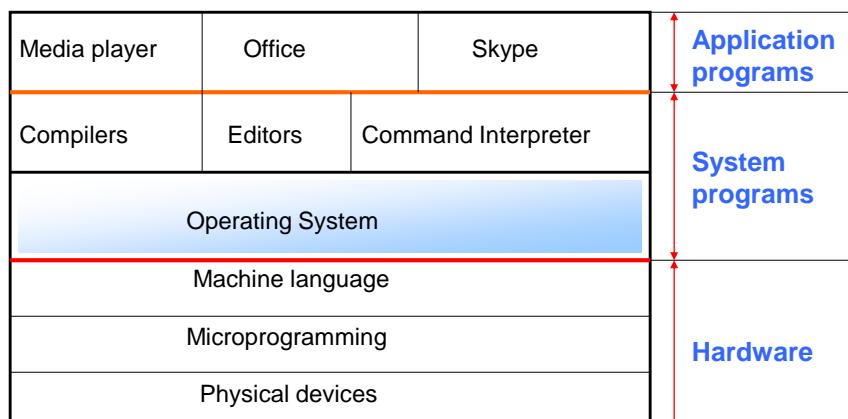
Layers and Views of a Computer System

- Computer Systems



3

Computer Systems



4

Operating Systems

- An operating system
 - a program that controls the execution of application programs and acts as an interface between the **user (program)** of a computer and **the computer hardware**.
- An operating system has four major components:
 - process management,
 - memory management
 - the file system
 - input/output
- The most common OS include
 - Windows, MacOS, and UNIX (like).

5

programs

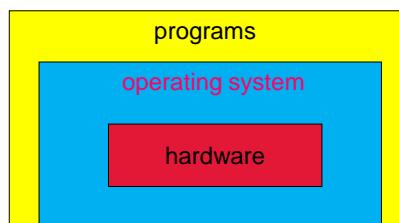
operating system

hardware

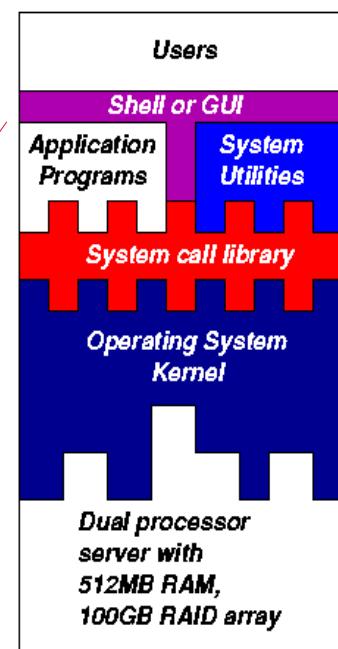
5

Operating systems

- General architecture
 - **Kernel**: deal with hardware
 - HW-independent expose to high level by **system call**
 - **Utilities**: small, come with OS
 - **Shell**: textual command-line interface



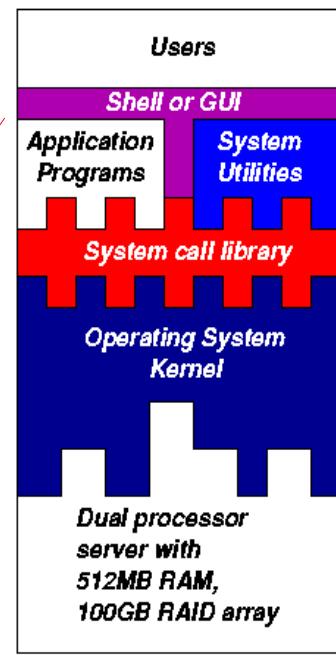
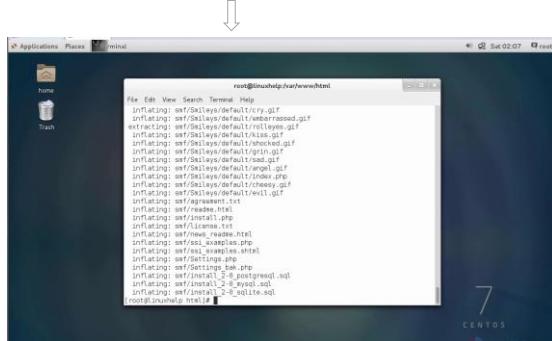
6



6

Operating systems

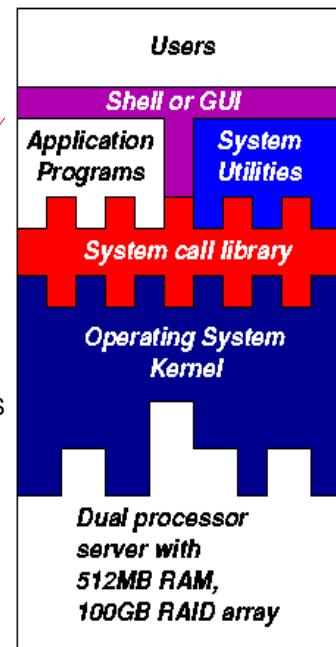
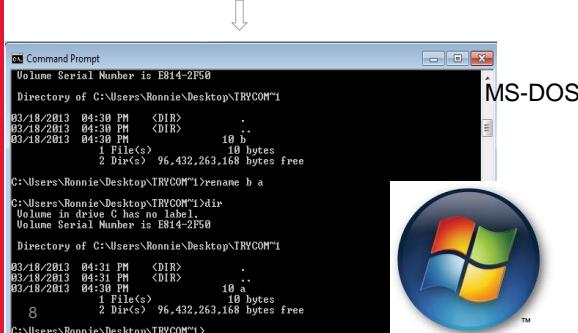
- General architecture
 - **Kernel**: deal with hardware
 - HW-independent expose to high level by **system call**
 - **Utilities**: small, come with OS
 - **Shell**: textual command-line interface



7

Operating systems

- General architecture
 - Kernel: deal with hardware
 - HW-independent expose to high level by **system call**
 - Utilities: small, come with OS
 - Shell: textual command-line interface

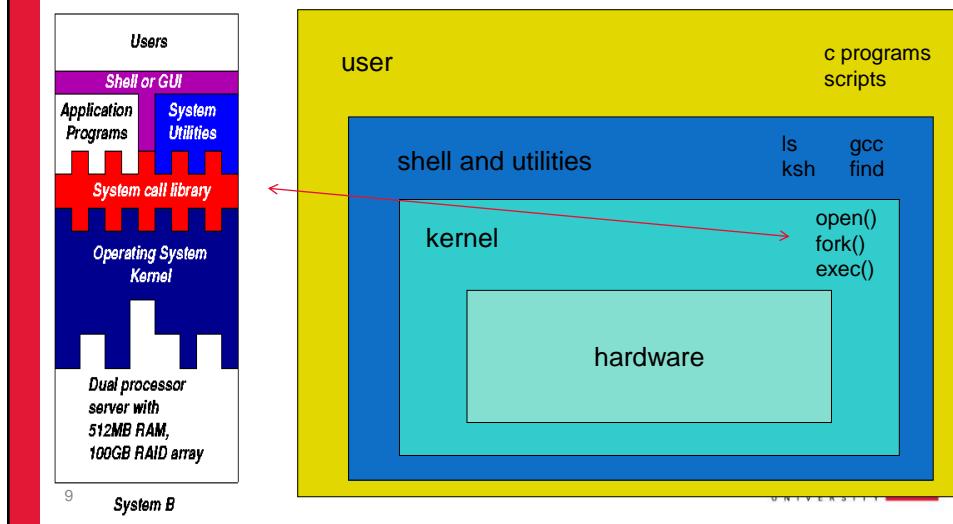


8

Unix System Structure

For your information

Unix is a popular operating system. It is most commonly used in backend applications, on servers, powerful workstations, etc. One of the most well-designed operating systems of its time.

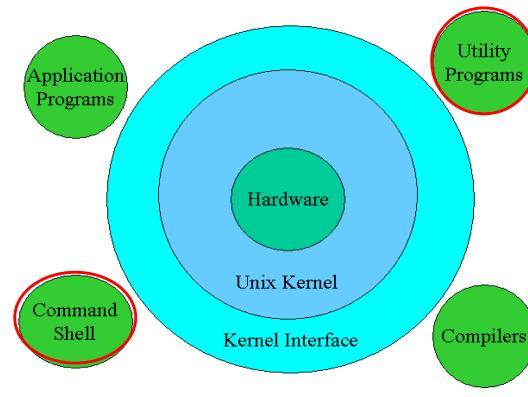


9

Unix System Structure

For your information

- Another picture



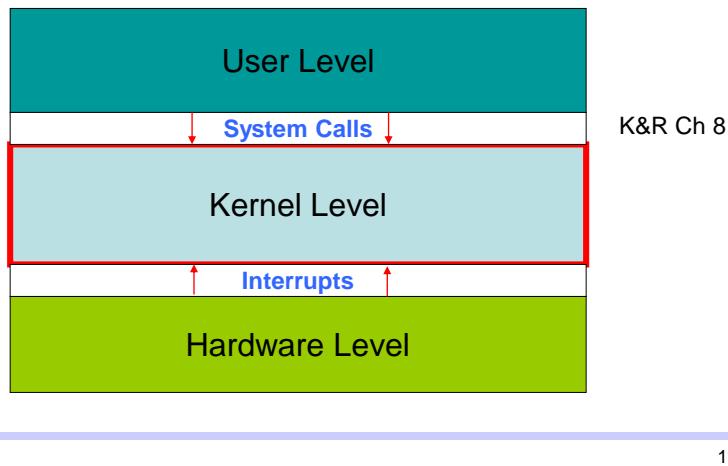
YORK U
UNIVERSITY

10

For your information

Kernel Interactions

- Processes accesses kernel facilities via [system calls](#)
- Peripherals communicate with the kernel via [hardware interrupts](#)

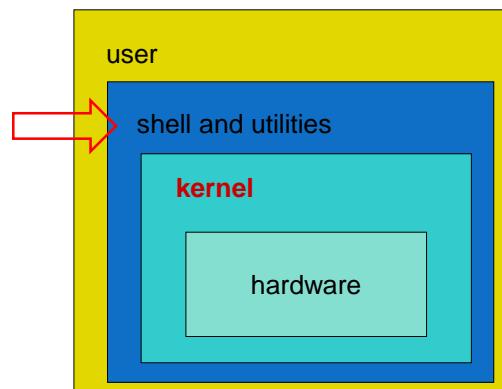


11

11

Shell and Utilities

- The rest of the operating system
- [Focus of the last 3 lectures of this course](#)

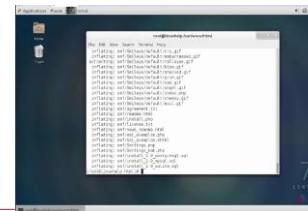


12

12

UNIX Shell (briefly)

- Shell is the (command line) **user interface** to the operating system
 - between you and the raw UNIX OS
 - when you log in, you interactively use the shell
- Functionality:
 - Execute other programs
 - Manage **files, processes**
- Command-line utilities\shell commands e.g., cd ls mkdir
- Scripting
 - A set of shell commands that constitute an executable program: **a script**
 - Batch file .bat in Windows



date
ls
.....

YORK U
UNIVERSITÉ
UNIVERSITY

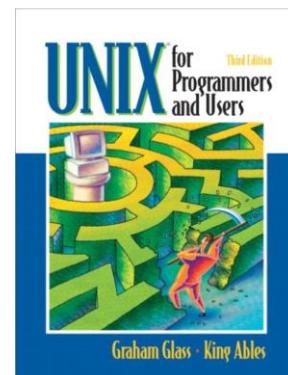
13

13

Contents

- Overview of UNIX
 - Structures
 - File systems
 - Pathname
 - Security
 - Process:
 - Exit code
 - Pipes
- Utilities/commands
 - Basic
 - Advanced
- Shell and shell scripting language

today

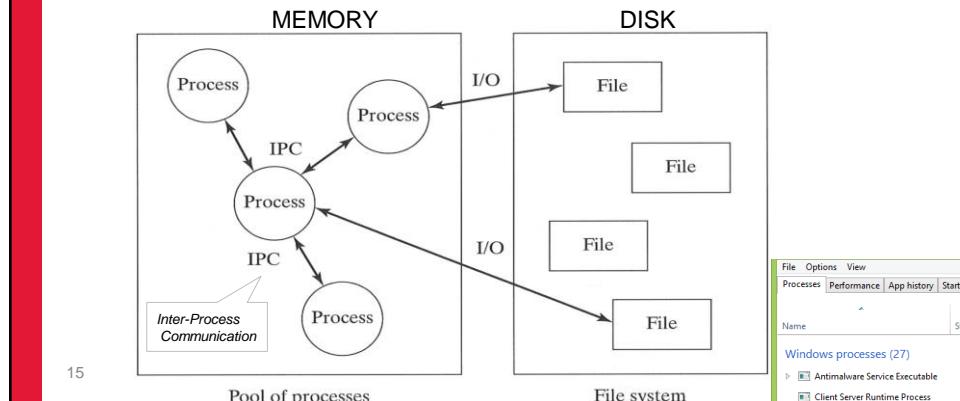


YORK U
UNIVERSITÉ
UNIVERSITY

14

Files and Processes

- A **file** is a collection of data that is usually stored on **DISK**
- When a program is invoked, it is **loaded from DISK into MEMORY**. When a program is running (in **MEMORY**), it is called a **process**.
- Most processes read and write data from files.

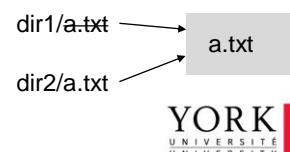


15

Unix File System

- Files are just sequences of bytes
- Directories are lists of files and other directories, along with their status:
 - creation date
 - permissions, etc.
- Each directory entry **links (points) to a file on the disk**
 - **moving a file does not actually move any data around.**
 - creates link in new location
 - deletes link in old location

Try to move (cut+paste) a 3G movie, see how quick it is



YORK **U**
UNIVERSITY

16

Unix file system

- Unix expands the usual definition of file to include anything from which data can be taken, sent., including io devices (kb, printer,)
 - In Unix, “**Everything is a file**”
- Four kinds of files
 - **Ordinary files** (regular files) hold info text/ binary files
 - **Special files** – (device files). Representing physical devices, key boards, terminals, printers and other peripherals.
 - **Directory files (directories)**. Hold other files and directories
 - **Link (pointer) to files.** hardlink, softlink

17

For your information



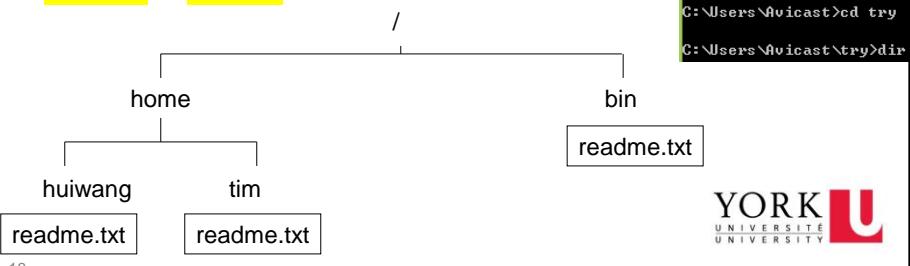
17

File Pathnames

- Two files in different directories can have the same name. We need **pathnames** to differentiate between files with the same names located in different directories.
- A **pathname** is a sequence of directory names that leads you through the hierarchy **from a starting directory to a target file**.

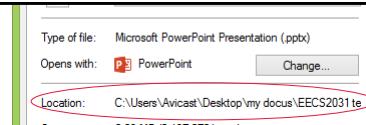
```
gcc /cs/dept/course/2018-19/W/2031Z/submit/lab3/cse12345/lab3A.c .
cat /home/tim/readme.txt
```

- **Absolute or Relative**



18

File: Absolute Pathnames

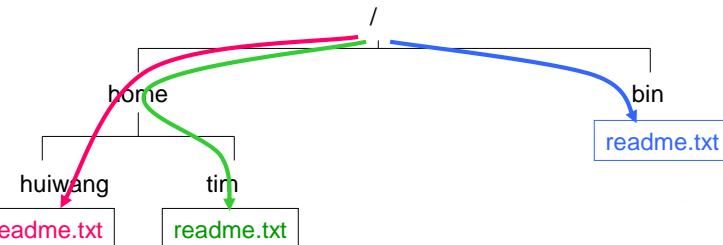


- A pathname starts from the root directory of file system is often termed an **absolute**, or **full** pathname.
- Valid from anywhere.

/home/huiwang/readme.txt ~/readme.txt

/home/tim/readme.txt

/bin/readme.txt



19 From anywhere, cat /home/tim/readme.txt

19

File: Relative Pathnames

- A process may also **unambiguously** specify a file by using a pathname **relative** to its current working directory.
- UNIX file system supports the following **special fields** that may be used when supplying a relative pathname:

Field	Meaning
.	current directory
..	parent directory

Same in
DOS

```
cd ..  
cat ./input.txt    cat input.txt  
./a.out < ../input.txt  
gcc .../a.c
```



20

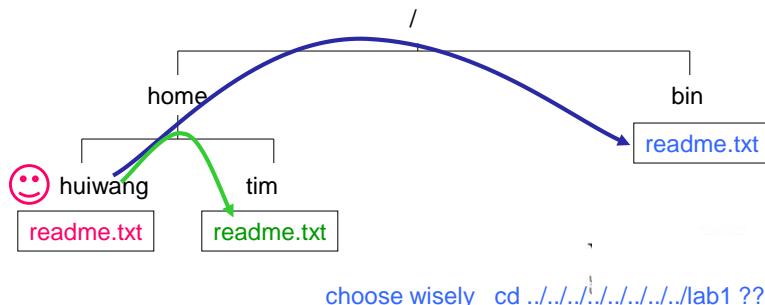
Relative Pathnames

- Relative Pathnames (from /home/huiwang)

cat readme.txt or ./readme.txt

cat ../tim/readme.txt

cat ../../bin/readme.txt



21

Links

For your information

- A *link* is a pointer to a file.
- In fact, in UNIX *all* filenames (directory entry) are just links to a file. Most files only have one link.

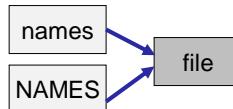
```
-rw-r--r--  1 jbond    cs          154 Feb  4 15:00 letter3
-rw-r--r--  1 jbond    cs           64 Feb  4 15:00 names
drwxr-xr-x  1 jbond    cs          512 Feb  4 15:00 secret/
```

- Additional links to a file allow the file to be shared.
- The `ln` command creates new links.

```
$ ln names NAMES
$ ls -l
total 8
-rw-r--r--  2 jbond    cs          64 Feb  6 18:36 NAMES
-rw-r--r--  1 jbond    cs          154 Feb  4 15:00 letter3
-rw-r--r--  2 jbond    cs           64 Feb  4 15:00 names
drwxr-xr-x  1 jbond    cs          512 Feb  4 15:00 secret/
```

22

Hard link



File: Unix Security

- Processes and files have an **owner** and may be protected against unauthorized access.
 - A set of users can form a **group**. A user can be a member of multiple groups.
 - Each user has a **primary (default) group**.
 - Your primary group is ‘ugrad’
 - also ‘submit’ and ‘labtest’
 - **red 302 % groups**

23



23

File and Directory Permissions

- UNIX provides a way to **protect files** based on users and groups.
 - Three types of permissions:
 - **read (r)** process may read contents of file
 - **write (w)** process may write contents of file
 - **execute (x)** process may execute file
 - Three sets/clusters of permissions:
 - permissions for **owner** -

The diagram shows a file permission string: -rwxr-xr--. Below it, three labels are positioned under the respective columns: 'owner' under the first column, 'group' under the second, and 'other' under the third. Each label has a red bracket underneath it, indicating the scope of the permissions listed above. The 'owner' row has three red ticks under 'rwx'. The 'group' row has two red ticks under 'rx'. The 'other' row has no red ticks under 'r--'.
 - permissions for **group**
 - permissions for **other**
 - Same types and sets of permissions as for files apply to **directories**:
 - **read** process may read the directory contents (i.e., list files)
 - **write** process may add/remove files in the directory
 - **execute** process may open files in directory or subdirectories

24

File Permissions (Security)

- File permissions are the basis for file security. They are given in three clusters.

1 - rw- r-x r-- 1 huiwang faculty 213 Jan 31 00:12 heart.final

User (owner)	Group	Others
r w -	r - x	r --

Each cluster of three letters has the same format:

Read permission	Write permission	Execute permission
r	w	x

e.g., webfile: others need to have r permission

submit dir: group need to have w permission

25

Permission examples

webfile: others must has r permission -rwxr-x---

The screenshot shows a browser window with the URL https://www.eecs.yorku.ca/course_archive/2018-19/S/2031/. The page displays a 'Forbidden' message: 'You don't have permission to access /course_archive/2018-19/S/2031/posts/lab7-fix.pdf on this server.' Below the message, it says 'Apache/2.4.39 (Unix) PHP/7.2.16 OpenSSL/1.0.2k Server at www.eecs.yorku.ca Port 443'.

submit directory: group must has w permission -rwxr-xr---

```
sh-4.2$ submit 2031 lab6 lab7D0.c
error: files may not be submitted for course 2031, assignment
lab6
Please contact your professor.
sh-4.2$
```

26 How to set/change permission?

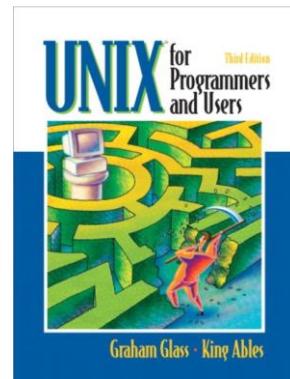


26

Contents

- Overview of UNIX
 - Structures
 - File systems
 - Pathname
 - Security
 - Process:
 - Exit code
 - Pipes
- Utilities/commands
 - Basic
 - Advanced
- Shell and shell scripting language

today



27

Processes

- Each command/utility involves a process
 - ls, cd, pwd, gedit ...
 - Unix can execute many processes simultaneously.
- When a process ends, there is a **return value** aka **exit code** associated with the process outcome
 - a non-negative integer. ≥ 0
 - 0 means **success**
 - anything > 0 represents various kinds of **failure**
 - The return value is passed to the parent process
 - Stored in system variable **\$?**

Opposite to C

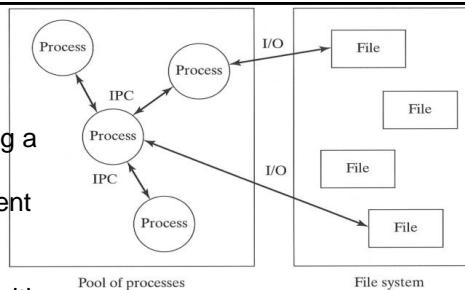
28

```
sh-4.2$ pwd
/cs/home/huiwang
sh-4.2$ echo $?
0
sh-4.2$ date
Sat Mar 30 09:15:52 EDT 2019
sh-4.2$ echo $?
0
sh-4.2$ cd xxx
sh: cd: xxx: No such file or directory
sh-4.2$ echo $?
1
sh-4.2$ ls xxx
ls: cannot access xxx: No such file or directory
sh-4.2$ echo $?
2
sh-4.2$
```

28

Process: Communication

- Processes can communicate using a number of means:
 - passing arguments, environment
 - **read/write regular disk files**
 - **exit values \$?**
 - inter-process communication with shared queues, memory and semaphores
 - signals
 - **pipes**
 - **sockets**
- A pipe is a one-way medium-speed data channel that allows two processes on the same machine to talk.
- If the processes are **on different machines** connected by a network, then a mechanism called a "socket" may be used instead. A socket is a **two-way** high-speed data channel.



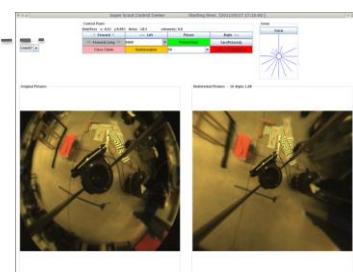
YORK U
UNIVERSITÉ
UNIVERSITY

29

29

Socket communication (on different machine)

Wireless network socket



Java, Python
Unix/Window/Mac



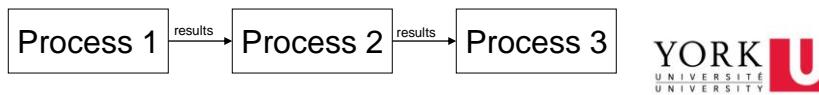
30

30

Process communication: Unix Pipes

- A special mechanism called a “pipe” built into the **heart** of UNIX to support cascading utilities.
- A pipe allows a user to specify that the **output of one process** is to be used as the **input to another process**.
- Two or more processes may be connected in this fashion, resulting in a “**pipeline**” of data flowing from the first process through to the last.

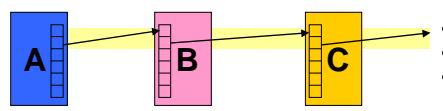
31



Process communication: Unix Pipes

- The nice thing about pipelines is that many problems can be solved by such an arrangement of processes.
- Each process in the pipeline performs a set of operations upon the data and then **passes the results** on to the next process for further processing.

32



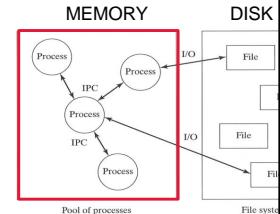
32

Pipeline Example

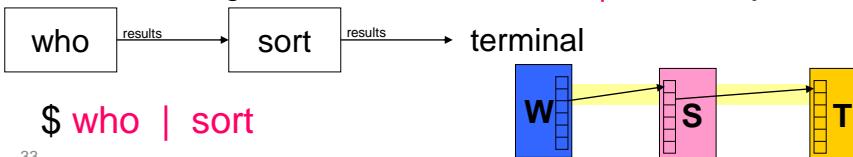
- A utility called **who** outputs an **unsorted list** of current users. Another utility called **sort** outputs a **sorted version of its input**.

\$ who

\$ sort input.txt # or sort < input.txt



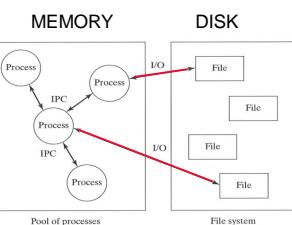
- These two utilities may be connected together with a “pipe” so that the output from **who** passes directly into **sort**, resulting in a sorted list of users. | does this job.



33

Pipe-Equivalent Communication Using a File

- Could we solve without using a pipe? **YES**.
- Run first program, save output into a disk file
- Run second program, using the file as input



- Disadvantages:**
 - Unnecessary use of the disk
 - Slower
 - Can take up a lot of space
 - Makes no use of multi-tasking
- Pipe is very similar, but does not involve the external device
 - all mechanisms stay in the realm of the operating system

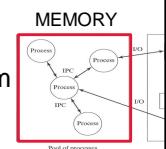


\$ who > tmp.txt
\$ sort < tmp.txt

34

\$ who | sort

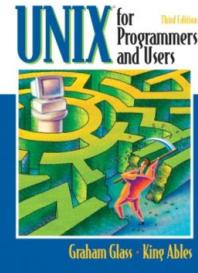
\$ ls | more



34

Contents

- Overview of UNIX
 - Structures
 - File systems
 - absolute vs relative pathname `../../input.txt`
 - security `-rwx--x--x`
 - Process:
 - has return value `0` (success) or `> 0` (sth wrong)
 - communication: **pipes** `who | sort` `ls | more`
 - Extra readings
- Utilities/commands
 - Basic
 - advanced
- Shell and shell scripting language



35

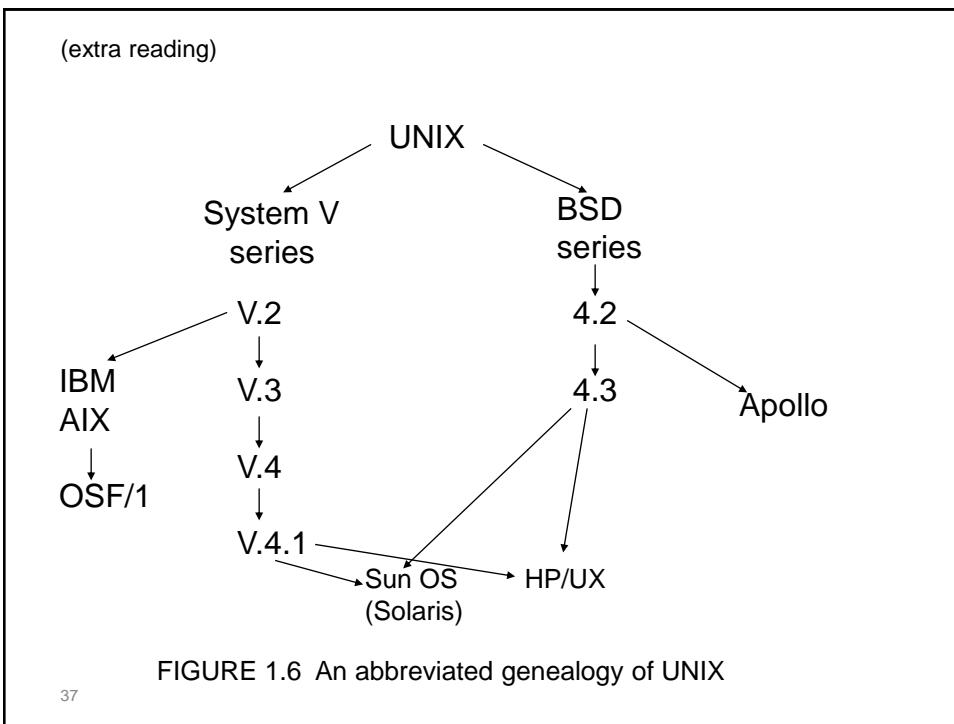
Unix Versions (extra reading)

- UNIX is a **fairly standard operating system**, with two main versions that are slowly merging into one.
- UNIX was created in **Bell Laboratories** and evolved from that into what is currently known as “**System V** UNIX.”
- The **university of California at Berkeley** obtained a copy of UNIX early on in its development and spawned another major version, known as **BSD (Berkeley Standard Distribution) UNIX**.
- **UNIX international**
 - AT&T, Sun Microsystems, --> **System V Release 4**.
- **Open Software Foundation**
 - IBM, Digital Equipment Corporation, Hewlett-Packard --> **BSD UNIX, called OSF/1**.

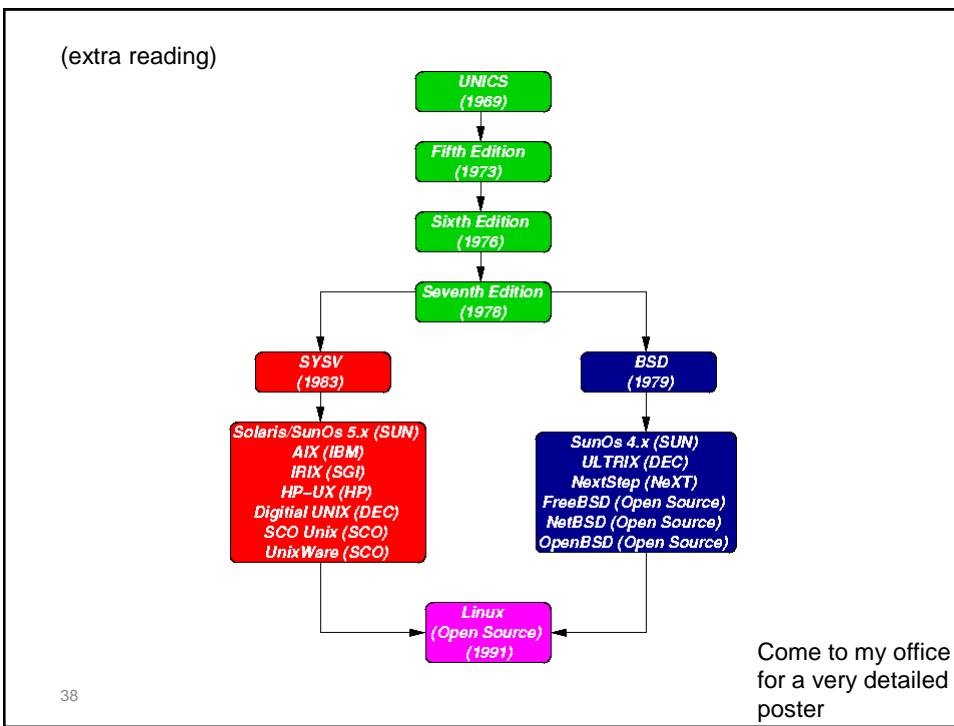


36

36



37



38

(extra reading)

Unix Standards

- Both groups tried to comply with a set of standards set by the [POSIX \(Portable Operating System Interface\)](#) committee
- Most of the best features of BSD UNIX have been rolled into most [System V-based versions of UNIX](#).
- UNIX is mostly [written in the C language](#), which makes it relatively [easy to port to different platforms](#).
- This feature is an important benefit and has contributed a great deal to [the proliferation and success of UNIX](#).



39

39

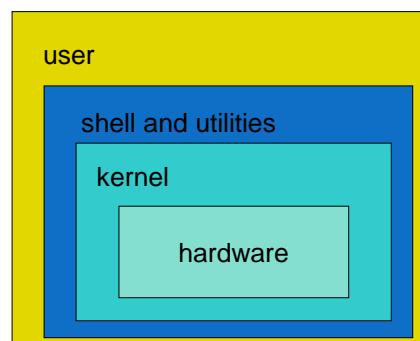
(extra reading)

Linux as an Unix-like OS

- Reimplementation. Adheres to the same POSIX standard as UNIX
 - Looks and acts alike
 - Bebian, Fedora, Red Hat, Ubuntu, SuSE, etc.



-
- Kernel
 - Shells and GUI
 - sh, bash, csh ...
 - KDE GNOME
 - System utilities
 - ls, cp, wc, more, grep, awk, sed ...
 - Application programs
 - Emacs, gcc, xfig, latex, soffice



40

40

(extra reading)

What is the difference between Linux and Unix?

- Ans 1: Linux is a unix-like kernel. It is open source.
- Ans2: To put it very generically, Linux is an operating system kernel, and UNIX is a certification for operating systems. The UNIX standard evolved from the original Unix system developed at Bell Labs. After Unix System V, it ceased to be developed as a single operating system, and was instead developed by various competing companies, such as Solaris (from Sun Microsystems), AIX (from IBM), HP-UX (from Hewlett-Packard), and IRIX (from Silicon Graphics). UNIX is a specification for baseline interoperability between these systems, even though there are many major architectural differences between them. Linux has never been certified as being a version of UNIX, so it is described as being "Unix-like." A comprehensive list of differences between Linux and "UNIX" isn't possible, because there are several completely different "UNIX" systems.

41



41

(extra reading)

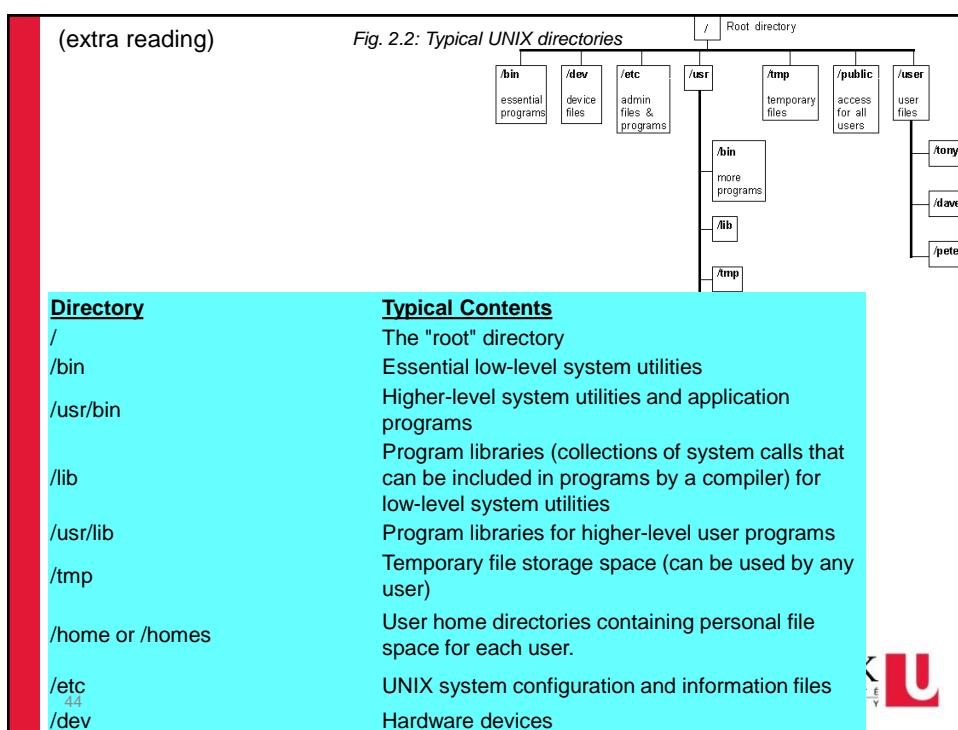
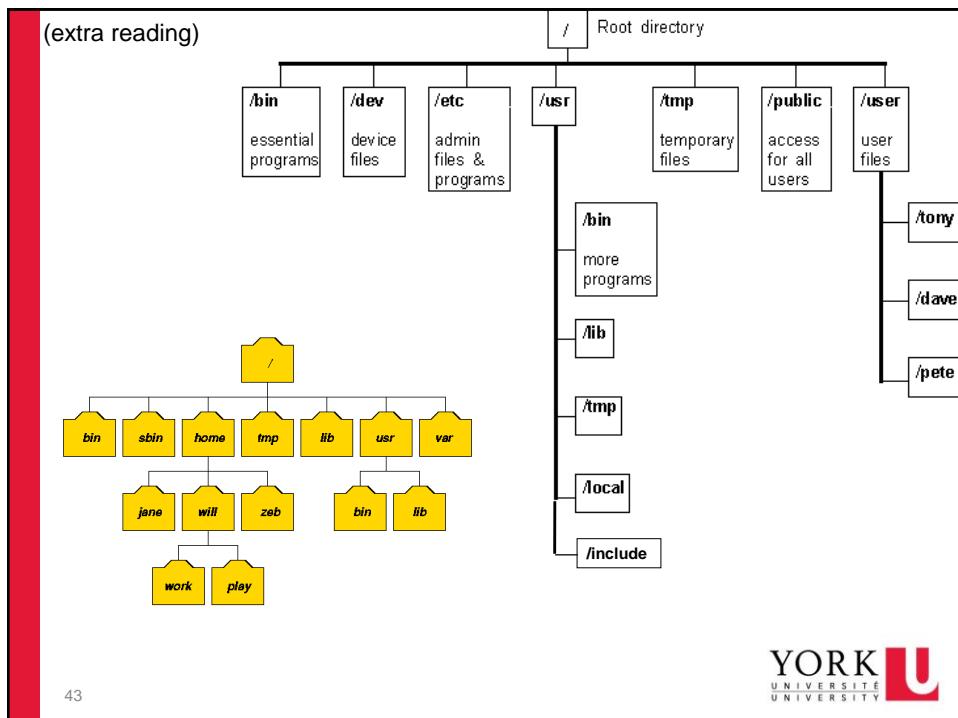
Unix file systems

- Unix expands the usual definition of file to include anything from which data can be taken, sent., including io devices (kb, printer,)
 - In Unix, “Everything is a file”
- Four kinds of files
 - Orindary files** (regular files) hold info text/ binary files
 - Special files** – (device files). Representing physical devices, keyboards, terminals, printers and other peripherals.
 - Directory files (directories).** Hold other files and directories
 - Link (pointer) to files** hardlink, softlink

42



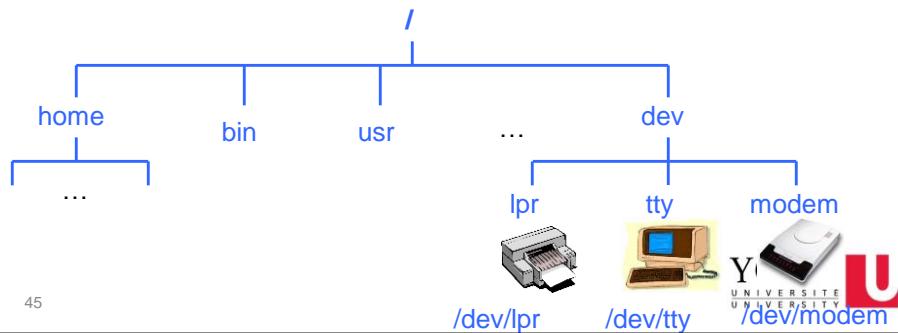
42



(extra reading)

Devices as Special Files

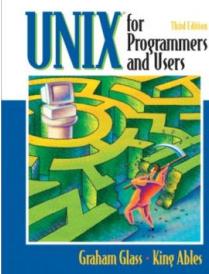
- Besides files, input and output can go from/to various hardware devices.
- Unix Philosophy: Treat these devices as **special files!**
- Terminals, printers, and other devices are **accessible in the same way as disk-based files.**



45

Contents

- Overview of UNIX
 - Structures
 - File systems
 - absolute and relative pathname `../../input.txt`
 - security `-rwx--x--x`
 - Process:
 - has return value `0` (success) or `> 0` (sth wrong)
 - communication: **pipes** `who | sort` `ls | more`
 - Extra readings
- Utilities/commands
 - Basic
 - advanced
- Shell and shell scripting language



46

Unix Utilities

- Standard UNIX comes complete with **at least 200 small utility programs**, usually including:
 - shells,
 - editors,
 - a C compiler,
 - matching with regular expressions,
 - searching,
 - a sorting utility,
 - software development tools,
 - text-processing tools, etc.

47



47

Basic utilities/commands

Introduces the following command-line utilities, listed in alphabetical order:

cancel	head	mv
cat	lp	newgrp
chgrp	lpr	page
chmod	lprm	passwd
chown	lpq	pwd
clear	lpstat	rm
cp	ls	rmdir
date	mail	stty
file	man	tail
groups	mkdir	tset
	more	wc

48



48

Basic utilities/commands

Introduces the following utilities, listed in groups:

General
man
clear
echo
date

Directory
pwd
ls
cd
mkdir
rmdir

File
cat
more
head tail
cp
mv
rm
wc
file
chmod
chgrp
newgrp
chown

File print
lp
lpr
lprm
lpq
lpstat



49

49

Running a utility/command

- To run a utility, simply enter its name at the prompt and press the Enter key.

- Up/down arrow for history
- Tab key for auto complete

```
$ date      # run the utility.  
Sun Jul 14 20:10:42 EDT 2019  
$ _
```



50

50

clear

- clears your screen

cls in DOS

echo

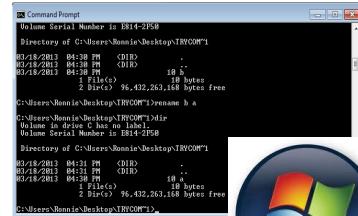
- print statement in UNIX/LINUX

- echo \$?

- echo hello

Same in DOS

```
sh-4.2$ echo hello
hello
sh-4.2$
```



51

51

man: online help

- All UNIX systems have a utility called **man** (short for **manual page**) that puts this information at your fingertips.
- The manual pages are **on-line copies of the original UNIX documentation**, which is usually divided into eight sections. They contain information about **utilities**, **system calls**, **file formats**, and **shells**.

man [section] word

man -k keyword

- The first usage of man **displays the manual entry** associated with **word**. If no section number is specified, the first entry that it finds is displayed.
- The second usage of man **displays a list of all the manual entries** that contain **keyword**.

52

52

Organization of the manual pages

- The typical division of topics in manual pages (**sections**) is as follows:

1. Commands and Application Programs.

2. System Calls

3. C Library Functions

4. Special Files

5. File Formats

6. Games

7. Miscellaneous

8. System Administration Utilities

% **man man**

```
MANUAL SECTIONS
The standard sections of the manual include:
 1  User Commands
 2  System Calls
 3  C Library Functions
 4  Devices and Special Files
 5  File Formats and Conventions
 6  Games et. Al.
 7  Miscellanea
 8  System Administration tools and Deamons
```

% **man 3 strlen**

53

53

Basic utilities/commands

Introduces the following utilities, listed in groups:

General

man

clear

echo

date

Directory

pwd

ls

cd

mkdir

rmdir

File

cat

more

head tail

cp

mv

rm

wc

file

chmod

chgrp

newgrp

chown

File print

lp

lpr

lprm

lpq

lpstat

54



54

pwd: Printing (present?) Working Directory

- To display your shell's [current working directory](#), use the [pwd](#) utility, which works like this:

```
login : huiwang
```

```
Password : .....
```

```
$pwd
```

```
/cs/home/huiwang
```

```
$cd a1    # cd ./a1
```

```
$pwd
```

```
/cs/home/huiwang/a1      # absolute pathname
```



55

55

Listing Contents of a Directory: ls

[dir in DOS](#)

```
ls [-adflsFR] { fileName }* { directoryName }*
```

- The [ls](#) lists all of the files and sub-directories in the current working directory in alphabetical order, [excluding files whose names start with a period](#).
- The [-l](#) option generates a long listing, including [permission flags](#), [the file's owner](#), and [the last modification time](#).
- The [-a](#) option causes [. . .](#) to be included in the listing.
- The [-d](#) option causes the details of the directories [themselves](#) to be listed, rather than their contents.
 - The [-S](#) option sorts the list on the size of entries.
 - The [-t](#) option sorts the list on the modification time.
 - The [-r](#) reverse the order of sorting

56

Directory Listing, an example

- Here's an example of the use of `Is`:

```
$ ls                                         # ls .  ls ./      list all files in current directory.  
a.out  heart.txt  
$ ls heart.txt  
heart.txt  
$ ls -l heart.txt                         # ls -l ./heart.txt    long listing of "heart.txt"  
1 -rw-r--r--  1 huiwang  faculty 106 Jan 30 19:46 heart.txt  
$  
  
hard-link count of the file  
username of the owner of the file  
group of the owner of the file  
size of the file, in bytes  
time that the file was last modified  
name of the file  
type and permission mode of the file
```



57

Directory Listing, an example

- More example of the use of **Is**:

```
$ ls a5          # ls ./a5  list all files in a5, a subdir of current directory.  
a.out          arrayAddressPPP.c  inputE2.txt  
arithmetic2017.c  inputA.txt
```

```
$ ls -l a5      # long listing of contents of a5
-rwx----- 1 huiwang faculty 7315 Mar 16 23:27 a.out
-rw------- 1 huiwang faculty 2210 Feb 20 15:40 arithmetic2017.c
-rw------- 1 huiwang faculty 1079 Feb 20 14:03 arrayAddressPPP.c
-rw------- 1 huiwang faculty    62 Feb 23 17:29 inputA.txt
-rw------- 1 huiwang faculty    50 Feb 23 19:56 inputE2.txt
```

```
$ ls -ld a5      # long listing of directory a5 itself  
drwxr--r-- 2 huiwang faculty 4096 Mar 16 23:27 a5
```

```
$ ls a5 -lSr ? # sort by size, same as ls -l -S -r or ls -IS -r  
$ ls lab6 -lt -r # sort by time. Who submitted lab6 in first/last minute?
```

58

Changing Directories: `cd`

Same in DOS

`cd [directoryName]`

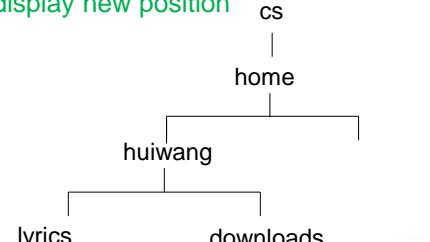
absolute or relative

- The following might be inconvenient; especially if we deal with large hierarchy:
`$ cat lyrics/heart.txt` # `./lyrics/heart.txt`
- Instead, change directory:
`$ cd lyrics` # change to directory `./lyrics`
`$ cat heart.txt`
- The `cd` shell command changes a shell's `current working directory` to be `directoryName`.
- If the `directoryName` argument is omitted, the shell is moved to its owner's home directory.
`$ cd # $cd ~`

59

Traversing Directories example

```
$ cd /cs/home/huiwang      # absolute or just cd
$ pwd                         # display where I am
/cs/home/huiwang
$ cd lyrics                   # ./lyrics move into the "lyrics" directory
$ pwd
/cs/home/huiwang/lyrics
$ cd ..                        # move up one level, back to huiwang
$ pwd                          # display new position
/cs/home/huiwang
$ cd downloads                 # ./downloads
$ pwd
/cs/home/huiwang/downloads
$ cd ../lyrics                 # go up one level, then into lyrics
$ pwd
/cs/home/huiwang/lyrics
$
```



60

60

Making Directory: `mkdir`

Same in DOS

`mkdir -p newDirectoryName`

- The `mkdir` utility creates a directory. The `-p` option creates any parent directories in the `newDirectoryName` pathname that do not already exist.
- If `newDirectoryName` already exists, an error message is displayed and the existing file is not altered

\$ `mkdir lyrics` # creates a directory called "lyrics".

\$ `ls -l` # check the directory listing to confirm

```
- rw-r--r-- 1 huiwang faculty 106 Jan 30 23:28 heart.ver1
drwxr-xr-x 2 huiwang faculty 512 Jan 31 19:49 lyrics/
```

\$ `mkdir -p study/programming/C/2019W/2031Z/lab2/`

61

Error without p, unless study/.../2031Z exists



Deleting a Directory: `rmdir`

Same in DOS

`rmdir {DirectoryName} +`

- The `rmdir` utility removes all of the directories in the list of directory names provided in the command.

- A directory must be empty before it can be removed.

\$ `rmdir lab5`

rmdir : lab5 : Directory not empty.

\$ _

- To (recursively) remove a directory and all of its contents, use the `rm` utility with the `-r` option

\$ `rm -r lab5`

62



Basic utilities

Introduces the following utilities, listed in alphabetical order:

General

man
clear
echo
date

Directory

pwd
ls
cd
mkdir
rmdir

File

cat
more
head tail
cp
mv
rm
wc
file
chmod
chgrp
newgrp
chown

Print

lp
lpr
lprm
lpq
lpstat

63



63

Creating a file with cat

Usage 1 of 3

`cat -n {fileName}*`

- The `cat` utility takes its input from standard input or from a list of files and displays them to standard output.
- `cat` is short for “concatenate” which means “to connect in a series of links.”
- By default, the standard input of a process is from the `keyboard` and the standard output is to the `screen`.

```
$ cat          # copy stdin input to stdout
hello
hello
^D          # tell cat that the end of input is reached.
$ _
$ cat > heart.txt      # store stdin input into a file called "heart.txt".
I hear her breathing,
I'm surrounded by the sound.
Floating in this secret place,
I never shall be found.
^D
$ _
```



64

Displaying a file with `cat`

Usage 2 of 3

- `cat` with the [name of the file](#) that you wanted to display:

```
$ cat heart.txt      # or cat < heart.txt  list contents of file heart.txt
```

I hear her breathing.

I'm surrounded by the sound.

Floating in this secret place,

I never shall be found.

```
$ _
```

- `cat` is good for [listing the contents of small files](#), but it [doesn't pause](#) between full screens of output.

- `more` is an alternative



65

65

Concatenate files with `cat`

Usage 3 of 3

- `cat` with [the name of the files](#) that you wanted to [concatenate](#) (display together):

```
$ cat heart.txt heart2.txt      # list the contents of both the files.
```

I hear her breathing.

I'm surrounded by the sound.

Floating in this secret place,

I never shall be found.

This is my heart

beating..

```
$ _
```

} heart.txt

} heart2.txt

- usually use redirection to create a new file concatenating contents of both the input files

```
$ cat heart.txt heart2.txt > heartNew.txt
```



66

66

Displaying a file: **more**

Same in DOS

more -f [+lineNumber] { fileName }*

- The **more** utility allows you to scroll a list of files, one page at a time.

- After each page is displayed, displays the message “--more– x%” to indicate that it's waiting for a command.

To display the next page, press the space bar.

To display the next line, press the Enter key.

To display the previous page, press ^B

To quit from **more**, press the “q” key.

- Try yourself:

```
$ ls -l /usr/bin > myLongFile      # myLongFile is a long file  
$ more myLongFile
```

```
67 $ ls -l /usr/bin | more          # use pipe
```



67

Displaying a file: **head** and **tail**

head -n { fileName }*

- The **head** utility displays the first n lines of a file. If n is not specified, it defaults to 10.

tail -n { fileName }*

- The **tail** utility displays the last n lines of a file. If n is not specified, it defaults to 10.

```
$ head -2 heart.txt                # list the first two lines.
```

I hear her breathing,
I'm surrounded by the sound.

```
$ tail -2 heart.txt                # list the last two lines.
```

Floating in this secret place,
I never shall be found.

Q: how to copy a file? How to copy a directory?

Q: how to rename a file/directory? Copy and remove old?



68

Copy a file/dir: `cp`

[copy in DOS](#)

- `cp location/fileName newLocation/newName`
 - if no `newName` => `cp location/fileName newLocation`
 - copy to `newLocation` with same name
 - if `newLocation` same as `location`
 - copy to same location with `newName`
- `cp -r location/dirName newLocation/newName`
 - copy directory.
- The `-i` option prompts you for confirmation if `newName` already exists so that you do not accidentally replace its contents.
- Now assume in the directory where the source file/dir exists
 - `cp fileName newLocation/newName`
 - 69 `cp -r dirName newLocation/newName`



69

Copy a file/dir: `cp`

[copy in DOS](#)

- `cp fileName newLocation/newName`
 - if no `newName` => `cp fileName newLocation`
 - copy to `newLocation` with same name
 - if no `newLocation` => `cp fileName newName`
 - copy to same (current) location with `newname`
- `cp -r dirName newLocation/newName`
 - if no `newName` => `cp -r dirName newLocation`
 - copy under `newLocation` with same name
 - if no `newLocation` => `cp -r dirName newName`
 - copy to same (current) location with `newName`

70



Copying Files: cp

cp, mv, a 3G movie, which is faster?

- cp actually does two things
 - It makes a physical copy of the original file's contents.
 - It creates a new label in the directory hierarchy that points to the copied file.

```
$ cp heart.ver1 heart.ver2 # copy ./heart.ver1 to ./heart.ver2
```

```
$ cp heart.ver1 ../../ # copy ./heart.ver1 to ../../ with same name
```

```
$ cp heart.ver1 ../../x.txt # copy ./heart.ver1 to ../../, name it "x.txt"
```

```
$ cp ../../x.txt . # ./ copy ../../x.txt to current dir, same name
```

```
$ cp ../../x.txt x2.txt # ./x2.txt copy to current dir, name it "x2.txt"
```

- The -r option causes any source files that are directories to be recursively copied, thus copying the entire directory structure.

```
$ cp -r dir1 dir2 # copy dir1, name it dir2
```

```
71 $ cp -r dir1 ../../lyrics/ # copy dir1 under lyrics, same name dir1
```

71

Renaming/Moving a file/directory: mv

- mv -i location/fileOrDirName newLocation/newName

- Move to newLocation with newName
- if no newName => move to newLocation with same name.
- if newLocation same as location =>
 - move to same location with newName
 - actually rename

- The -i option prompts you for confirmation if newName already exists so that you do not accidentally replace its contents.

- Now assume in the directory where fileOrDirName exist

- mv fileOrDirName newLocation/newName

72



Renaming/Moving a file/directory: mv

- **mv fileOrDirName newLocation/newName**

- Move to **newLocation** with **newName**
- if no **newName** => **mv fileOrDirName newLocation**
 - move to **newLocation** with same name **move in DOS**
- if no **newLocation** => **mv fileOrDirName newName**
 - move to same (current) location with **newName**
 - actually **rename** **rename in DOS**

73

Renaming/Moving Files: mv **rename in DOS**

- Here's how to **rename** file/dir using the first form of the mv utility:

```
$ ls -l  
1 -rwxr--r-- 1 huiwang faculty 409 Mar 10 20:57 heart.txt  
1 drwxr--r-- 1 huiwang faculty 4096 Mar 16 23:27 lyrics  
  
$ mv heart.txt heart2.txt # rename to "heart2.txt".  
$ ls  
heart2.txt lyrics  
$ mv lyrics lyrics2019 # rename a directory  
$ ls  
heart2.txt lyrics2019
```

- We will see other use (**move** files)



74

Moving Files

move in DOS

```
$ ls -l
```

```
1 -rwxr--r-- 1 huiwang faculty 409 Mar 10 20:57 heart.txt  
1 drwxr--r-- 1 huiwang faculty 4096 Mar 16 23:27 lyrics
```

```
$ mv heart.txt lyrics      # ./lyrics move into "lyrics"  
                                (same name)
```

```
$ ls
```

```
lyrics/          # "heart.txt" has gone.
```

```
$ ls lyrics      # list the "lyrics" directory.
```

```
heart.txt        # "heart.txt" has moved.
```

```
$ mv lyrics/heart.txt .  # move back (with same name)
```

```
$ mv heart.txt lyrics/heart.ver2 #move and rename
```

No real data movement, just entry / link
/ pointer switch



75

Deleting files: rm

del in DOS

- The rm utility removes a file's label from the hierarchy.

```
rm -fir {fileName}*  
75
```

- If the filename doesn't exist, an error message is displayed.
- The -i option prompts the user for confirmation before deleting a filename. It is a very good idea to use this option
- The -f option inhibits all error messages and prompts. It overrides the -i option
This is dangerous!
- If fileName is a directory, the -r option causes all of its contents, including subdirectories, to be recursively deleted.
 - Really used for deleting directories.

76

Removing Directories with Files

- The **-r** option of **rm** can be used to delete the “lab1” directory and all of its contents with just one command:

```
$ rm a.out      # remove a file.
```

```
$ rm -r lab1    # recursively delete directory.
```

```
$ rm -f -r *
```



77

cp vs mv

- Copy (copy+paste) and move (cut+paste) a 3G movie, which is faster?
- Below will both rename file1 to file2, what is the difference?

```
cp file1 file2
```

```
rm file1
```

```
mv file1 file2
```



78

Counting Lines, Words and Chars in Files: **WC**

```
wc -lwc {fileName}*  
79
```

- The wc utility **counts the number of lines, words, and/or characters** in a list of files.
- If no files are specified, standard input is used instead.
- **-l** option requests a **line** count,
- **-w** option requests a **word** count,
- **-c** option requests a **character** count.
- If no options are specified, then all three counts are displayed.
- A **word** is defined by a sequence of characters surrounded by tabs, spaces, or new lines.



79

Counting Lines, Words and Characters in Files: **WC**

- For example, to count lines, words and characters in the “heart.txt” file, we used:

```
$ wc heart.txt      # obtain a count of the number of lines,  
                      words, and characters.  
9     43    213  heart.txt
```

-
- Given class list file “EECS2031A”, in which each line represents one student. How many students are there in the class? Let’s do it

```
$ wc -l EECS2031A  
$ cat EECS2031A | wc -l      # another way, using pipe
```

- How many people are currently logging onto eecs server?

```
80 $ who | wc -l      # using pipe
```

80

File Attributes

- We used `ls` to obtain a long listing of “heart.txt” and got the following output:

```
$ ls -l heart.txt
1 -rw-r--r-- 1 huiwang faculty 213 Jan 31 00:12 heart.txt
$ ls -ld lyrics
1 drwxr-xr-- 1 huiwang faculty 533 Jan 31 00:12 lyrics
$ _
```



81

81

```
$ ls -l heart.txt
1 -rw-r--r-- 1 huiwang faculty 213 Jan 31 00:12 heart.txt
```

File Attributes

Field #	Field value	Meaning
1	1	the number of blocks of physical storage occupied by the file
2	-rw-r--r--	the type and permission mode of the file, which indicates who can read, write, and execute the file 
3	1	the hard-link count
4	huiwang	the username of the owner of the file
5	faculty	the group name of the file
6	213	the size of the file, in bytes
7	Jan 31 00:12	the time that the file was last modified
8	heart.final	the name of the file

82

82



File Attributes

- **File Types**

- Field 2 describes the file's type and permission settings.

```
1 -rw-r--r-- 1 huiwang faculty 213 Jan 31 00:12 heart.final
```

- The first character of field 2 indicates the type of file, which is encoded as follows :

character	File Type
-	regular file
d	directory file
b	buffered special file(such as a disk drive)
c	unbuffered special file(such as a terminal)
l	symbolic link
p	pipe
s	socket

83



83

Determining Type of a File: `file`

```
file fileName(s)
```

- The `file` utility attempts to describe the contents of the `fileName` argument(s), including the language in which any of the text is written.
- not reliable; it may get confused.

```
$ file heart.txt      # determine the file type.  
heart.txt: ASCII text  
$ file lab5B.c  
lab5B.c: C source, ASCII text  
$ file a.out  
a.out: ELF 64-bit LSB executable, x86-64, version 1 (SYSV) .....
```

84

File Permissions (Security)

- File permissions are the basis for file security. They are given in three clusters.

1 - rw- r-x r-- 1 huiwang faculty 213 Jan 31 00:12 heart.final

User (owner)	Group	Others
r w -	r - x	r --

clusters

Each cluster of three letters has the same format:

Read permission	Write permission	Execute permission
r	w	x

e.g., webfile: others need to have r
submit dir: group need to have w

How to set/change



85

Permission examples

webfile: others must have r permission -rwxr-x---

The screenshot shows a browser window with the URL https://www.eecs.yorku.ca/course_archive/2018-19/S/2031/. The page displays a "Forbidden" message: "You don't have permission to access /course_archive/2018-19/S/2031/posts/lab7-fix.pdf on this server." Below the message, it says "Apache/2.4.39 (Unix) PHP/7.2.16 OpenSSL/1.0.2k Server at www.eecs.yorku.ca Port 443".

submit directory: group must have w permission -rwxr-xr---

```
sh-4.2$ submit 2031 lab6 lab7D0.c
error: files may not be submitted for course 2031, assignment
lab6
Please contact your professor.
sh-4.2$
```

86 How to set/change permission?



chmod



86



Change File Permissions: `chmod`

Only owner and admin can change

`chmod -R change[, change]* {fileName }+`

- The `chmod` utility changes the **modes (permissions)** of the specified files according to the change parameters, which may take the following forms:

`clusterSelection + newPermissions` (add permissions)
`clusterSelection - newPermissions` (subtract permissions)
`clusterSelection = newPermissions` (assign permissions absolutely)

where **clusterSelection** is any combination of:

u (user/owner)
g (group)
o (others)
a (all)



`rw- r-x r--`
u g o
a

newPermissions is any combination of
r (read) **w** (write) **x** (execute)

- ⁸⁷ The `-R` option recursively changes the modes of the files in directories.

87

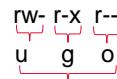
`r-w- r-x r--`
u g o
a

Changing File Permissions: examples

Requirement	Change parameters
<u>Add group write permission</u>	<code>g+w</code>
<u>Remove group write permission</u>	<code>g-w</code>
<u>Remove other's read and write permission</u>	<code>o-rw</code>
<u>Add execute permission for user, group, and others.</u>	<code>a+x u+x,g+x,o+x</code>
<u>Give the group read permission only.</u>	<code>g=r</code>
<u>Add write permission for user, and remove group read permission.</u>	<code>u+w, g-r</code>
<u>Give the other read and execute permission</u>	<code>o=wx</code>

88

Changing File Permission: examples



- Here's an example of how to set these permissions:

```
$ ls -l lab4.pdf  
1 -rw-r----- 1 huiwang faculty 213 Jan 31 00:12 lab4.pdf  
$ chmod o+r lab4.pdf  
$ ls -l lab4.pdf  
1 -rw-r--r-- 1 huiwang faculty 213 Jan 31 00:12 lab4.pdf  
$ chmod a+x lab4.pdf  
$ ls -l lab4.pdf  
1 -rwxr-xr-x 1 huiwang faculty 213 Jan 31 00:12 lab4.pdf  
  
$ ls -ld 2031      # list attributes of directory 2031.  
1 drwxr-xr-x 45 huiwang faculty 4096 Apr 29 14:35  
$ chmod o-rx 2031      # update permissions. other -rx  
$ ls -ld 2031      # confirm.  
1 drwxr-x--- 45 huiwang faculty 4096 Apr 29 14:35  
$
```



89

Changing Permissions Using Numbers

- The **chmod** utility allows you to specify the new permission setting of a file as 3 octal numbers (0~7).
- Each octal digit (0~7) represents a permission triplet.

binary 1/0 1/0 1/0

r w x

For example, if you wanted a file to have the permission settings of

rwx r-x --- # owner:rwx, group r x → chmod u=rwx, g=rx

then the octal permission setting would be **750**, calculated as follows:

	User	Group	Others
setting	rwx	r-x	---
binary	111	101	000
octal	7	5	0

90

90

Changing File Permissions Using Octal Numbers

- The octal permission setting would be supplied to `chmod` as follows:

```
$ chmod 750 lab4.pdf      # or chmod u=rwx, g=rx lab4.pdf
$ ls -l lab4.pdf          # confirm.
1 - rwx r-x ---  45 huiwang faculty 4096 Apr 29 14:35 lab4.pdf
$ _
```

7	5	0
111	101	000
rwx	r - x	---



91

91

Changing Permissions Using Octal Numbers

- The `chmod` utility allows you to specify the new permission setting of a file as an octal number.

rwx	7	Read, write and execute	111
rw-	6	Read, write	110
r-x	5	Read, and execute	101
r--	4	Read,	100
-wx	3	Write and execute	011
-w-	2	Write	010
--x	1	Execute	001
---	0	no permissions	000



chmod u=rwx, g=rwx, o=rw	chmod 775		
chmod u=rwx, g=rx, o=	chmod 750		
chmod u=rw, g=r, o=r	chmod 644		
chmod u=rw, g=r, o=	chmod 640		
chmod u=rw, go=	chmod 600		
chmod u=rwx, go=	chmod 700		

92

An example: setting up submit directory using `chmod`

- <https://wiki.eecs.yorku.ca/dept/tdb/services:submit:submit-setup>

The screenshot shows a sidebar with links like Department of Electrical Engineering & Computer Science, Technical Database, News, Departmental Services, E-Mail, Lab Schedules, Login and Remote Access, Operating System, Policies and Procedures, Printing, Scanning, Software, Web Publishing, and Wiki Publishing. The main content area has a header 'SUBMIT DIRECTORY SETUP' and a breadcrumb trail 'Technical Database » Departmental Services » Submit » Submit Directory Setup'. It contains instructions for setting up a submit directory, a code block for creating the directory structure, and a note about removing submissions.

In order to setup a submit directory for your course:

- The course directory must be under /eecs/course.
- In the course directory, create a directory called "submit". That should be accessible by everyone.
- Under the submit folder, create one directory per assignment. The assignment directory must be writable by group, not by "other".

For example, to setup a submit directory for course 1021 and assignment a1, use the following command:

```
% mkdir /eecs/course/1021 <- this is only necessary if you haven't created it before  
% chmod 755 /eecs/course/1021  
% mkdir /eecs/course/1021/submit  
% chmod 755 /eecs/course/1021/submit  
% mkdir /eecs/course/1021/submit/a1  
% chgrp submit /eecs/course/1020/submit/a1  
% chmod 770 /eecs/course/1021/submit/a1
```

If you no longer wish to allow submissions for an assignment (e.g. past a due date) then remove the directory:

```
% chmod g-w /eecs/course/1021/submit/a1
```

- [Also next page](#)

[Let's do it](#)

UNIVERSITY OF YORK

93

Example of `chmod`, `chgrp`

- Create a submission directory for weekly lab
 - `mkdir lab8`
 - `chgrp submit lab8 # change group to 'submit'`
 - `chmod 770 lab8 # or chmod u=rwx, g=rwx lab8`

```
$ mkdir lab8  
$ ls -ld lab8  
1 drwx----- 2 huiwang faculty 4096 Jul 11 16:39 lab8
```

```
$ chgrp submit lab8  
$ ls -ld lab8  
1 drwx----- 2 huiwang submit 4096 Jul 11 16:39 lab8
```

```
$ chmod 770 lab8  
$ ls -ld lab8  
1 drwxrwx--- 2 huiwang submit 4096 Jul 11 16:39 lab8  
7 7 0
```

- After due time, close the submission

```
$ chmod g-w lab8 # chmod 750 lab8  
1 drwxr-x--- 2 huiwang submit 4096 Jul 11 16:39 lab8
```



94

Basic utilities Summary

Introduces the following utilities, listed in alphabetical order:

General

man
clear
echo
date

Directory

pwd
ls
cd
mkdir
rmdir

File

cat
more
head tail
cp
mv
rm
wc
file
chmod
chgrp
chown
newgrp

File

lp
lpr
lprm
lpq
lpstat

95

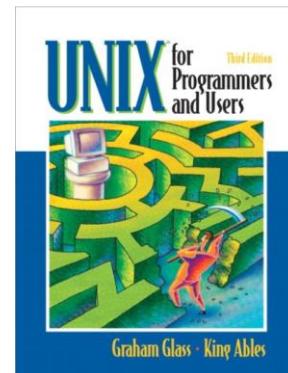


95

Contents

- Overview of UNIX
 - Structures
 - File systems
 - Pathname
 - Security
 - Process:
 - Pipes
 - Exit code
- Utilities/commands
 - Basic
 - Advanced
- Shell and shell scripting language

today



96