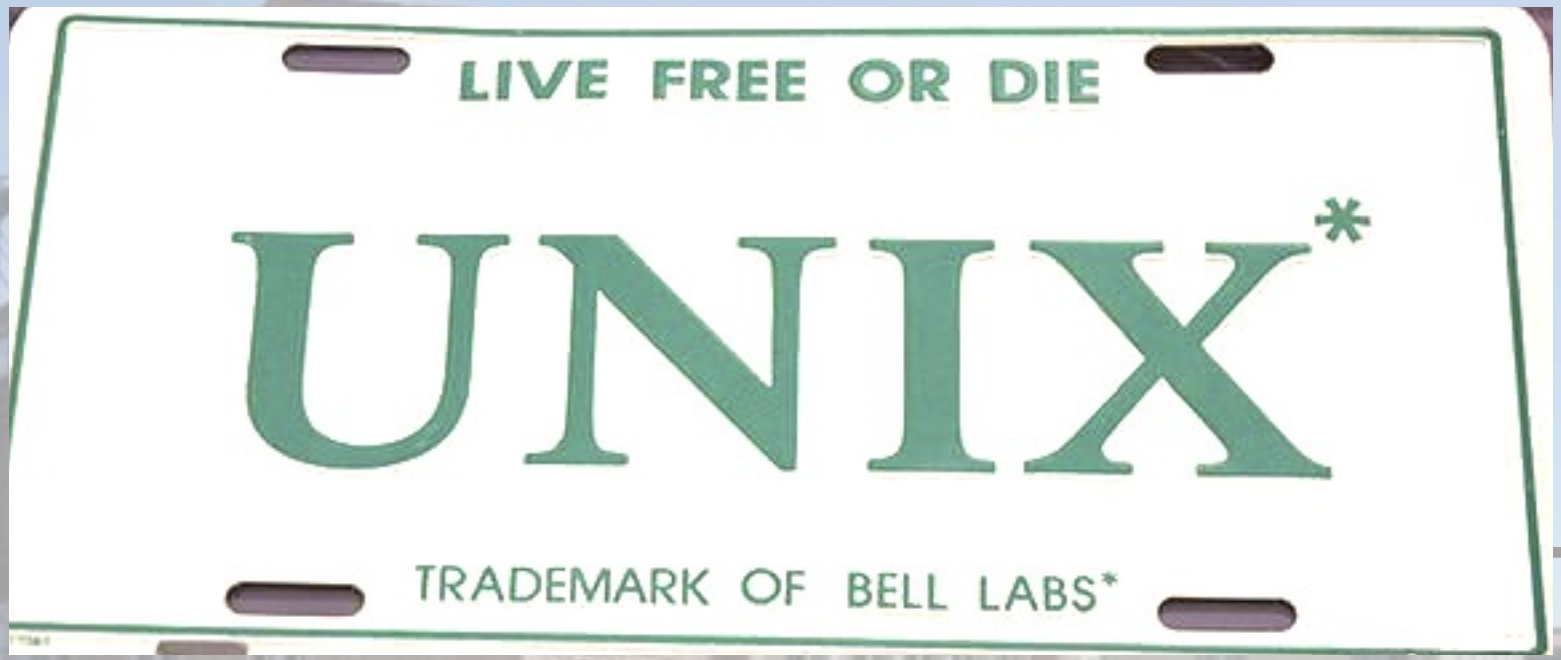


CSCI 3308

Liz Boese

What is Unix?

- Unix is an operating system
  - sits between the hardware and the user/applications
  - provides high-level abstractions (e.g., files) and services (e.g., multiprogramming)

- Linux:
  - a "Unix-like" operating system: user-level interface very similar to Unix
  - code base is different from original Unix code

MEMORABLE LINUX MILESTONES

CELEBRATING 20 YEARS OF LINUX

1991 — LINUS TORVALDS POSTS FAMOUS MESSAGE - *"HELLO EVERYBODY OUT THERE..."* - AND RELEASES FIRST LINUX CODE

1992 — LINUS LICENSES LINUX UNDER THE GPL, AN IMPORTANT DECISION THAT WILL CONTRIBUTE TO ITS SUCCESS IN THE COMING YEARS

1993 — SLACKWARE BECOMES FIRST WIDELY ADOPTED DISTRIBUTION

1996 — LINUS VISITS AQUARIUM, GETS BIT BY A PENGUIN AND CHOOSES IT AS LINUX MASCOT

1998 — TECH GIANTS BEGIN ANNOUNCING PLATFORM SUPPORT FOR LINUX

1999 — RED HAT GOES PUBLIC

2003 — IBM RUNS FAMOUS LINUX AD DURING THE SUPERBOWL

2005 — LINUS APPEARS ON THE COVER OF BUSINESSWEEK WITH A STORY THAT HAILS LINUX AS A BUSINESS SUCCESS

2007 — THE LINUX FOUNDATION IS FORMED TO PROMOTE PROTECT AND STANDARDIZE LINUX LINUS IS A FELLOW

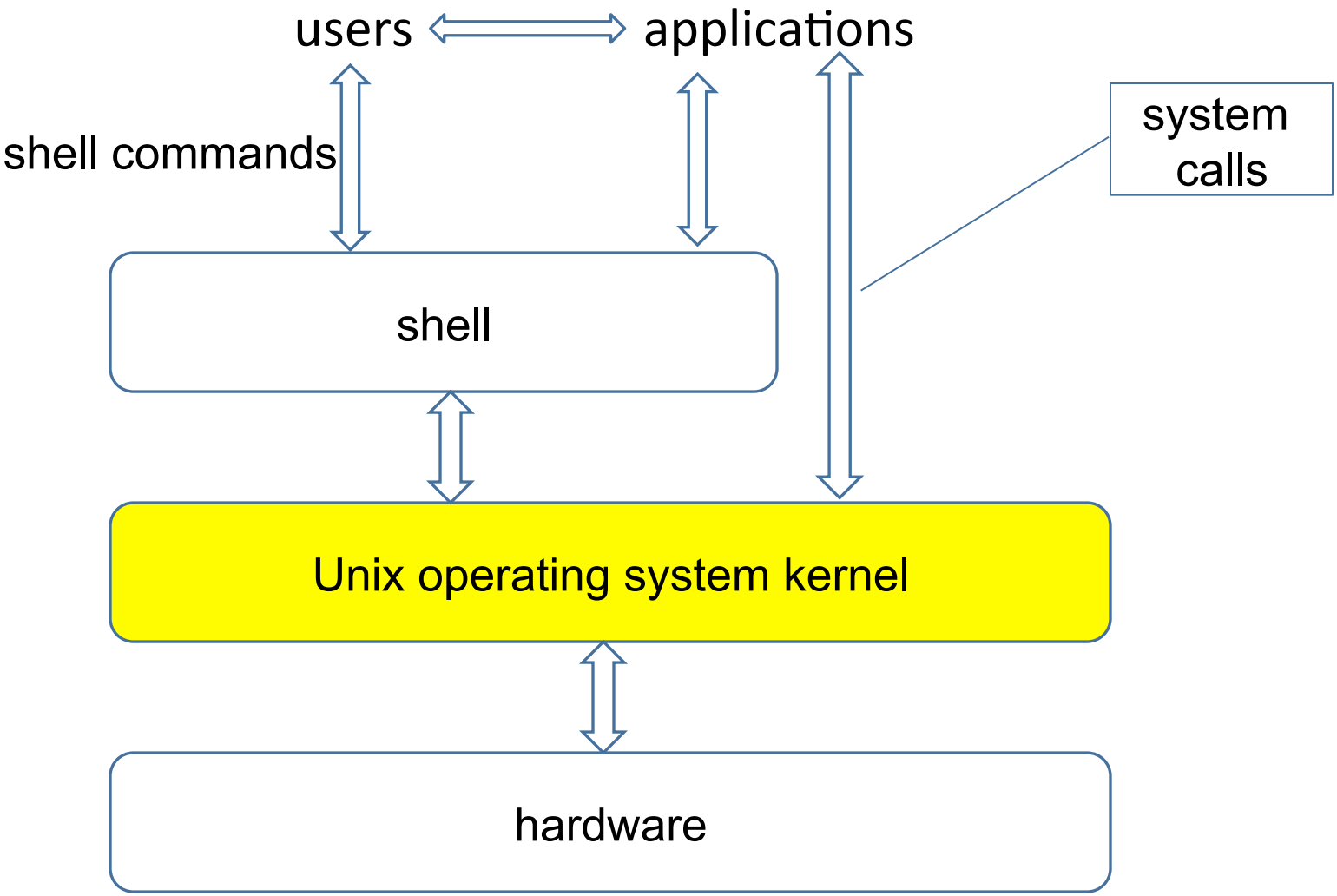
2010 — THE LINUX-BASED ANDROID OS OUTSHIPS ALL OTHER SMARTPHONE OSES IN THE U.S. AND CLIMBS TO DOMINANCE

2011 — LINUX TURNS 20 AND POWERS THE WORLD'S SUPERCOMPUTERS, STOCK EXCHANGES, PHONES, ATMS, HEALTHCARE RECORDS, SMART GRIDS, THE LIST GOES ON

20 YEARS OF LINUX 1991 - 2011

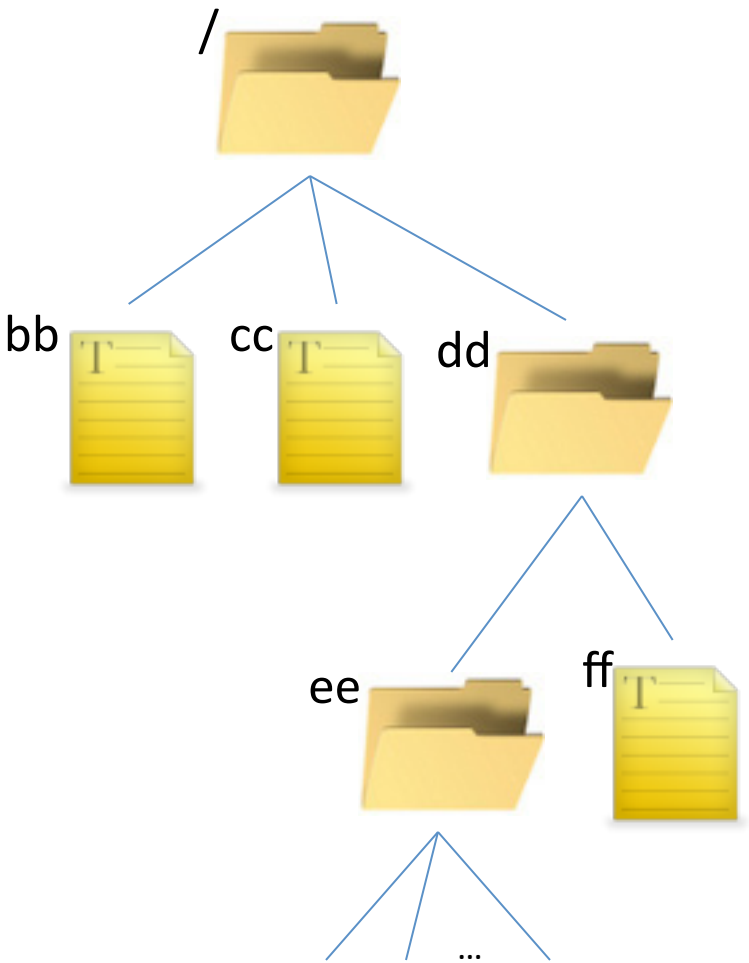
THE LINUX FOUNDATION
http://www.linuxfoundation.org/

Layers of a Unix system



The file system

- A *file* is basically a sequence of bytes
- Collections of files are grouped into *directories* (≈ folders)
- A directory is itself a file
  - file system has a hierarchical structure (i.e., like a tree)
    - the root is referred to as "/"

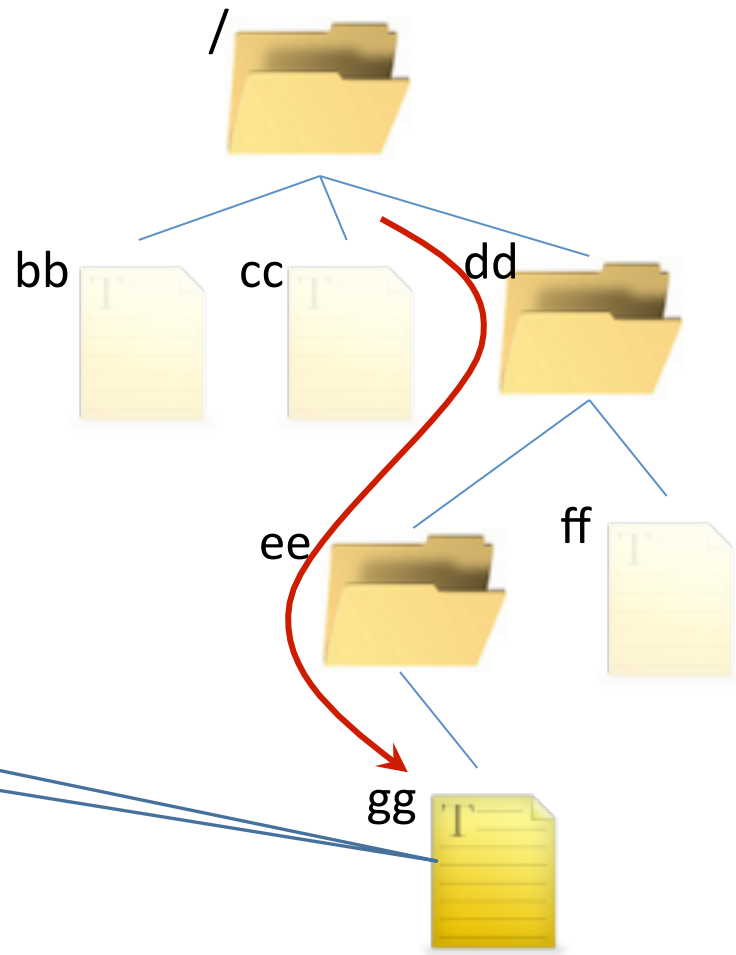


"Everything is a file"

- In Unix, everything looks like a file:
  - documents stored on disk
  - directories
  - inter-process communication
  - network connections
  - devices (printers, graphics cards, interactive terminals, …)
- They are accessed in a uniform way:
  - consistent API (e.g., read, write, open, close, …)
  - consistent naming scheme (e.g., /home/debray, /dev/cdrom)

Referring to files: Absolute Paths

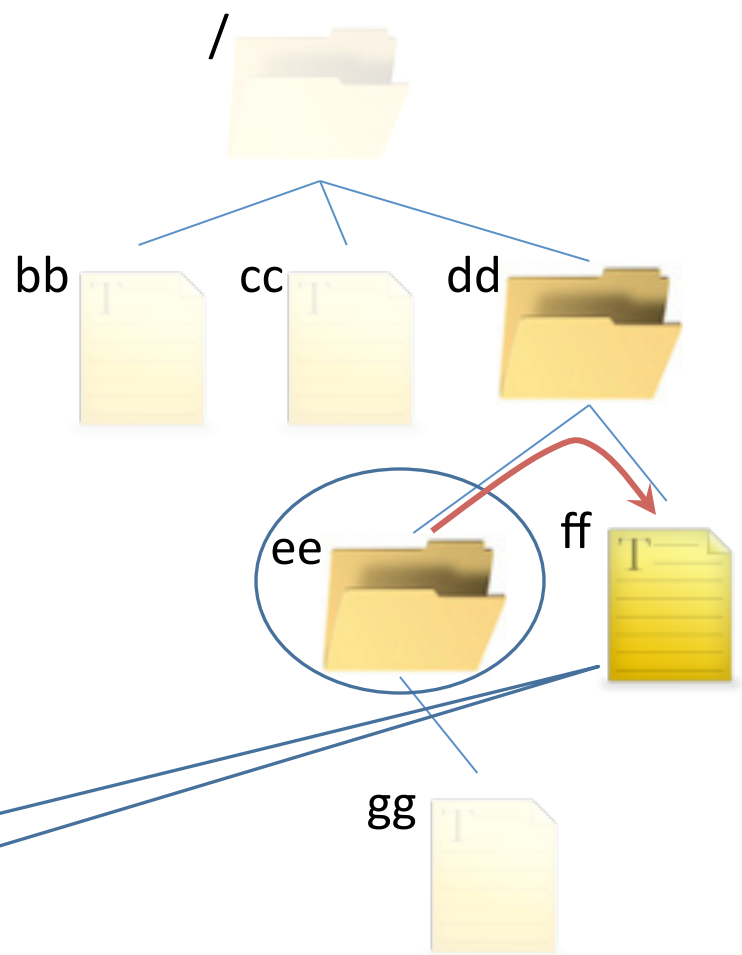
- An *absolute path* specifies how to get to a file starting at the file system root
  - list the directories on the path from the root ("/"), separated by "/"



absolute path: **/dd/ee/gg**

Referring to files: Relative Paths

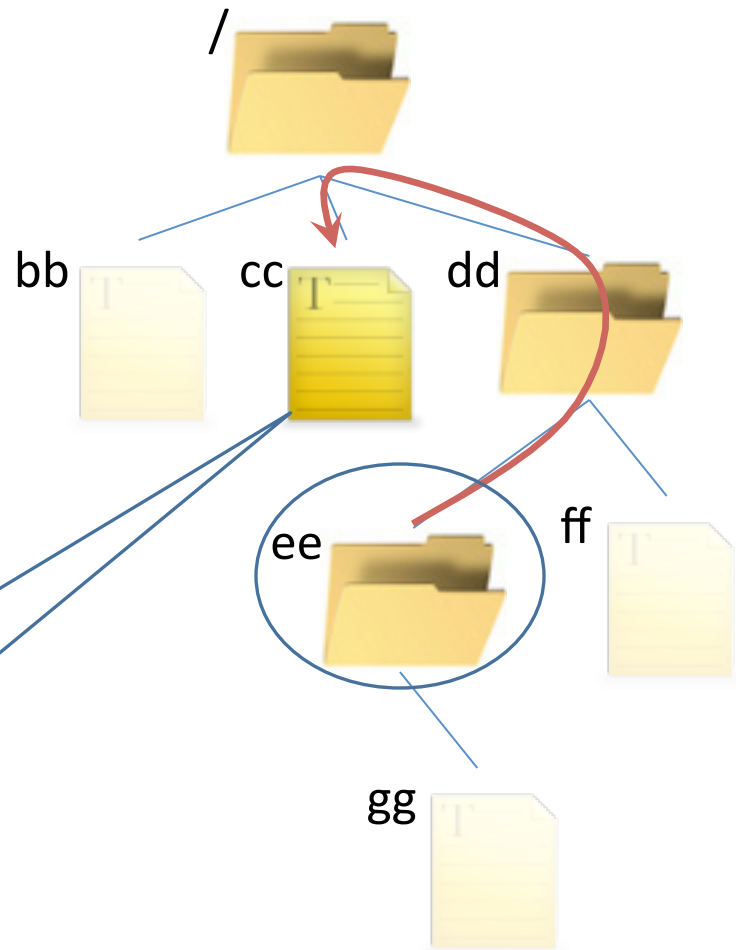
- Typically we have a notion of a "*current directory*"
- A *relative path* specifies how to get to a file starting from the current directory
  - '..' means "move up one level"
  - '.' means current directory
  - list the directories on the path separated by "/"



Example:
ff relative to **ee** is: **../ff**

Referring to files: Relative Paths

- Typically we have a notion of a "*current directory*"
- A *relative path* specifies how to get to a file starting from the current directory
  - '..' means "move up one level"
  - '.' means current directory
  - list the directories on the path separated by "/"



Example:
cc relative to **ee**
is: **../../cc**

File completion

- Type part of name of a command or filename, press <tab> and will auto-complete the name for you.
- Arrow keys to go back through history of commands you typed in.

history

- Display a history of recently used commands
- `history`
  - all commands in the history
- `history 10`
  - last 10
- `history -r 10`
  - reverse order
- `!!`
  - repeat last command
- `!n`
  - repeat command **n** in the history where n is a # *(type history to see the #s)*
- `!-1`
  - repeat last command = `!!`
- `!-2`
  - repeat second last command
- `!ca`
  - repeat last command that begins with 'ca'
- `!$`
  - Repeat last *argument* from last command

Home directories

- Each user has a "home directory"
  - specified when the account is created
  - given in the file **/etc/passwd**

- When you log in, your current directory is your home directory
  - can then start a *shell* and issue commands

A shell is just an interpreter for Unix commands

- Notational shorthand:
  - One's own home directory: ~
  - some other user **joe**'s home directory: **~joe**

Input and output

- Data are read from and written to i/o *streams*
- There are three predefined streams:

  stdin : "standard input" - usually, keyboard input

  stdout : "standard output" - usually, the screen

  stderr : "standard error" - for error messages (usually, the screen)

Other streams can be created using system calls (e.g., to read or write a specific file)

Processes

- Programs are executed via *processes*
  - a process is the unit of execution
  - consists of:
    - » the code that is executed
    - » the data this code manipulates
- Different processes execute concurrently
  - each process has its own address space, stdin, stdout, etc.
  - their execution is managed by the operating system
- Common tasks are carried out using a set of system-provided programs called *commands*
  - the shell is also a system-provided program

Unix Commands

- Each command performs [variations of] a single task
  - "options" can be used to modify what a command does
  - different commands can be "glued together" to perform more complex tasks
- Syntax:

  command options arguments

  Examples:

Command	Options	Arguments
pwd		
cd		/home/debray
ls	-a -l	
ls	-al	/usr/local

Options can (usually) be combined together:

Unix Commands

- Each command performs [variations of] a single task
  - "options" can be used to modify what a command does
  - different commands can be "glued together" to perform more complex tasks
- Syntax:

  command *options* *arguments*

Not always required: may have default values

Examples:

typical defaults:
- input: stdin
- output: stdout
- directory: current

Command	Options	Arguments
pwd		
cd		/home/debray
ls	-a -l	
ls	-al	/usr/local

Examples of Unix commands I

- Figuring out one's current directory: **pwd**

- Moving to another directory: **cd** *targetdir*

  Examples:

**cd / **	move to the root of the file system
cd ~ (also: just "**cd**" by itself)	move to one's home directory
cd /usr/local/src	move to /usr/local/src
cd ../..	move up two levels

mkdir

Create a directory

```
mkdir newdir
```

Examples of Unix commands II

- Command: **ls** — *lists the contents of a directory*
  - Examples:

ls	list the files in the current directory ⚠ *won't show files whose names start with '.'*
ls /usr/bin	list the files in the directory /usr/bin
ls -l	give a "long format" listing (provides additional info about files)
ls -a	list all files in the current directory, including those that start with '.'
ls -al /usr/local	give a "long format" listing of all the files (incl. those starting with '.') in /usr/local

ls

access permissions

```
drwxrwxr-x   3 root     bin          8704 Sep 23  2004 /usr/bin
-r-xr-xr-x   1 bin      bin          9388 Jul 16  1997 /usr/bin/cat
```

owner | group | size | last-modified time | file name

# of hard links

file type

-		normal file
d		directory
l	(*ell*)	symbolic link

File access permissions

```
--- = 0
--x = 1
-w- = 2
-wx = 3
r-- = 4
r-x = 5
rw- = 6
rwx = 7
```

```
$ ls -ld /usr/bin /usr/bin/cat
drwxrwxr-x   3 root     bin          8704 Sep 23  2004 /usr/bin
-r-xr-xr-x   1 bin      bin          9388 Jul 16  1997 /usr/bin/cat
```

access permissions for others (**o**)

access permissions for group (**g**)

access permissions for owner (**u**)

r	read
w	write
x	execute *(executable file)* enter (directory)
-	no permission

Changing file access permissions

Command:

chmod $\boxed{who}\pm\boxed{what}$ $file_1$ $file_2$ … $file_n$

$\in \{r, w, x\}$

$\in \{a, u, g, o\}$

Example:

chmod u-w foo	remove write permission for user on file foo
chmod g+rx bar	give read and execute permission to group for bar
chmod o-rwx *.doc	remove all access permissions for "other users" (i.e., not owner or group members) for *.doc files
chmod a+rw p*	give read and write permission to everyone for all files starting with p
chmod 754 file	

Combining commands

- The output of one command can be fed to another command as input.
  - Syntax: *command*$_1$ | *command*$_2$

"pipe"

Example:

ls	lists the files in a directory
more *foo*	shows the file *foo* one screenful at a time
ls | more	lists the files in a directory one screenful at a time

How this works:

- **ls** writes its output to its **stdout**
- **more**'s input stream defaults to its **stdin**
- the pipe connects **ls**'s stdout to **more**'s stdin
- the piped commands run "in parallel"

Finding out about commands I

Figuring out which command to use

apropos *keyword*

man –k *keyword*

"searches a set of database files containing short descriptions of system commands for keywords"

- Helpful, but not a panacea:
  - depends on appropriate choice of keywords
    - » may require trial and error
  - may return a lot of results to sift through
    - » pipe through **more**

Finding out about commands II

Figuring out how to use a command

man *command*

"displays the on-line manual pages"

- Provides information about command options, arguments, return values, bugs, etc.

Example: “man ls”

```
hed: /home/debray
LS(1)                          User Commands                          LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about  the FILEs (the current directory by default).  Sort entries alphabetically if none of
       -cftuvSUX nor --sort.

       Mandatory arguments to long options are mandatory for short options too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..

       --author
              with -l, print the author of each file

       -b, --escape
              print octal escapes for nongraphic characters

       --block-size=SIZE
              use SIZE-byte blocks

       -B, --ignore-backups
              do not list implied entries ending with ~

       -c     with -lt: sort by, and show, ctime (time of last modification of file status information) with -l:  show
              ctime and sort by name otherwise: sort by ctime

       -C     list entries by columns

       --color[=WHEN]
              control whether color is used to distinguish file types.  WHEN may be `never', `always', or `auto'
:
```

items within square brackets are optional

Some other useful commands

- **wc** [*file*]
  - » *word count*: counts characters, words, and lines in the input

```
wc file
wc –l file
wc –w file
```

- **grep** *pattern* [*file*]
  - » select lines in the input that match *pattern*

```
grep public *java
grep include controller.cpp
grep TODO src/*
ls | grep –i main
```

Some other useful commands

- **Cat**

  cat filename

- **Head**

  `head filename`

  » show the first 10 lines of the input

  `head –n filename`

  » show the first *n* lines of the input

- **Tail**

  `tail filename`

  » show the last 10 lines of the input

  `tail –n filename`

  » show the last *n* lines of the input

- **Copy**

  cp $file_1$ $file_2$

- **Move or rename**

  mv $file_1$ $file_2$

I/O Redirection

- Default input/output behavior for commands:
  - **stdin**: keyboard; **stdout**: screen; **stderr**: screen
- We can change this using I/O redirection:

cmd < *file*	redirect **cmd**'s stdin to read from *file*
cmd > *file*	redirect **cmd**'s stdout to *file*
cmd >> *file*	append **cmd**'s stdout to *file*
cmd &> *file*	redirect **cmd**'s stdout and stderr to *file*
$\mathbf{cmd}_1$ | $\mathbf{cmd}_2$	redirect $\mathbf{cmd}_1$'s stdout to $\mathbf{cmd}_2$'s stdin

diff

Compare two files

- `diff file1 file2`
- `sdiff file1 file2`

cut

- cut
  - `cut -c2  test.txt`
  - `cut -c1-3 test.txt | sort`
  - `cut -d' ' -f2 test.txt`
  - `cut -d':' -f1,6 /etc/passwd`

find

- Find a file in a directory tree.
  (period means to start in the current directory)
- `find . -name filename -print`

script

- Writes a log (a typescript) of whatever happened in the terminal to a file.
- `script [file]`
- `script`
  - all log is saved into a file named typescript
- `script file`
  - all log is saved into a file named file
- To exit logging, type:
  - `exit`

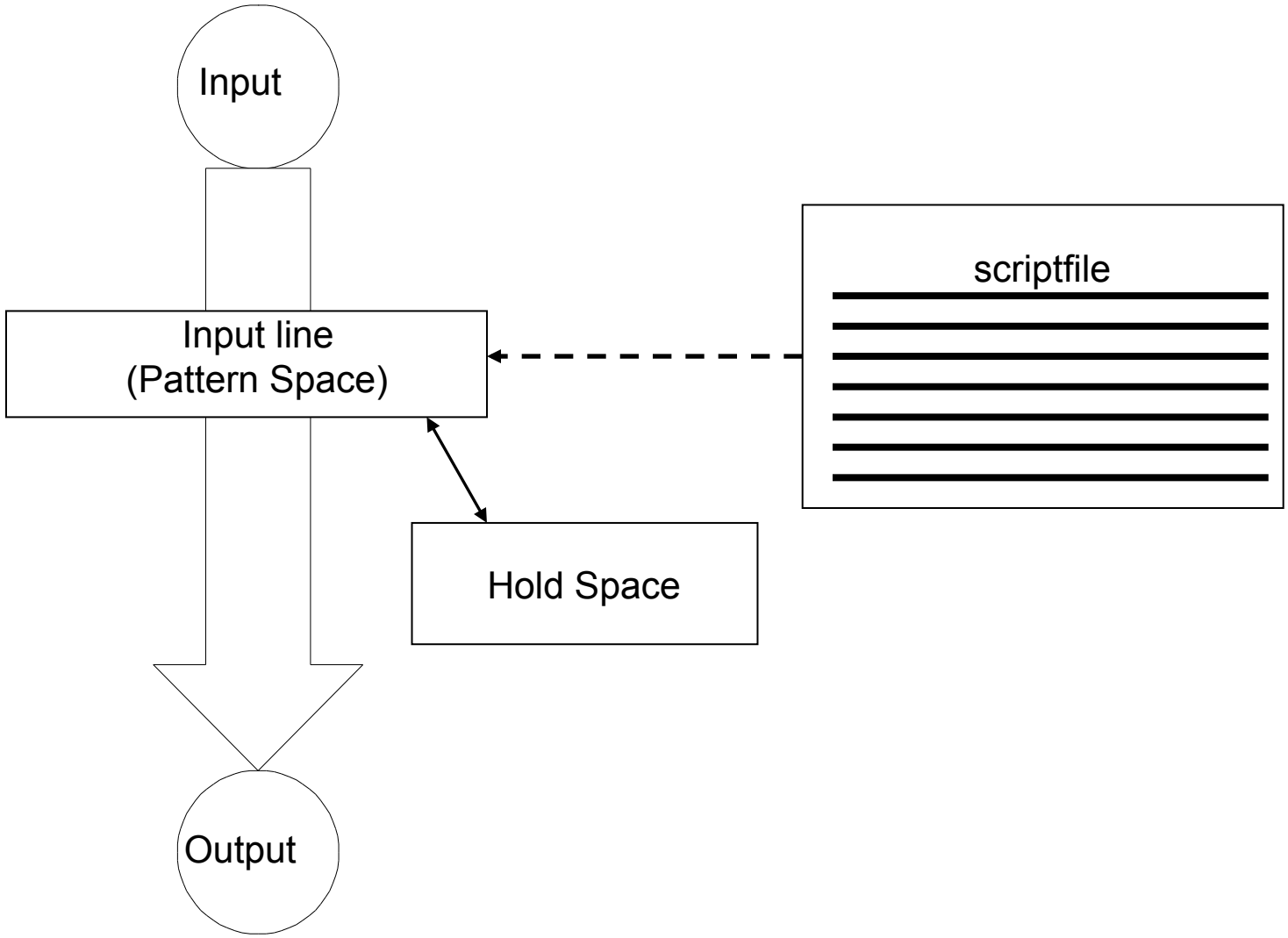
Serguei A. Mokhov,
mokhov@cs.concordia.ca

SED

sed: string replacement

- Line-oriented tool for pattern matching and replacement (stream editor)
- Not really a programming language
- Good for
  - Apply same change to lots of source files
    - » E.g., remove comments in all student submission files before submitting to grade script
  - Edit files too large for interactive editing
- Filter, i.e., does not modify input file *(-i option will modify the file)*

Sed Architecture



sed Syntax

- sed [-n] [-e] [‘command’] [file…]
- sed [-n] [-f scriptfile] [file…]
- -n - only print lines specified with the print command (or the ‘p’ flag of the substitute (‘s’) command)
- -f scriptfile - next argument is a filename containing editing commands
- -e command - the next argument is an editing command rather than a filename, useful if multiple commands are specified

Scripts

- As each line of the input file is read, sed reads the first command of the script and checks the address against the current input line:
  - if there is a match, the command is executed
  - if there is no match, the command is ignored
  - sed then repeats this action for every command in the script file

- When it has reached the end of the script, sed outputs the current line (pattern space) unless the -n option has been set

sed description

- pattern a text → add to output
- address s /regex/replacement/
- address d → delete line
- Delete lines 1-10: `sed '1,10d‘ file`
- Print lines 1-10: `sed -n '1,10p‘ file`
- Delete comments: `sed '/^#/d‘ file`
- Print only matching:

```
sed -n -e '/regexp/p’ file
sed -n -e ‘/^[fs]/p’ file
```

- Delete all blank lines: `sed ‘/^$/d’ file`

sed

Example:
Which command prints all the lines in the file, replace all occurrences of “ive” with “iff”?

- `sed -e 's/ive/iff/p' data`
- `sed -e 's/ive/iff/' data`
- `sed -n -e 's/ive/iff/p' data`
- `sed -n -e 's/ive/iff/' data`
- `sed  's/ive/iff/' data`
- `sed  's/ive/iff/g' data`

```
one
two
three  three four five
four
five 1 3 five fiver thrive
#six
#seven hive hiver shiver
#eight
nine
ten
```

AWK

- Awk basic syntax:

```
awk '/searchPattern/ {Actions}' InputFile
```

- Awk without pattern-matching

```
awk ‘{Actions}' InputFile
```

- Awk with pre- and post- processing

```
awk  ‘BEGIN {FS=“:”}
      {Actions}
      END {print “Done”}' InputFile
```

- Awk commands in a file

```
awk –f awkcommands.awk InputFile
```

awk

- Special-purpose language for line-oriented pattern processing
- pattern {action}
- Patterns = boolean combinations of regular expressions and relational expressions
- action =
  - if (conditional) *statement* else *statement*
  - while (conditional) statement
  - break
  - continue
  - variable=expression
  - print expression-list

awk

- Examples:
  - Print lines longer than 72 characters: ($0 for the line)

    `length($0) > 72`

```
awk 'length($0) > 72' data
```

  - print first two fields in opposite order

    `{ print $2,$1 }`

```
awk '{print $2, $1}' data
```

vs

```
awk '{print $2 $1}' data
```

awk

- **`awk –f program.awk < input > output`**
- Example File program.awk

```
BEGIN {FS=":"} {print $2 $1}
```

awk examples

- Add up first column, print sum and average

```
{s += $1 }
END {print “sum is”, s, “average is”, s/NR}
```

- Print all lines between start/stop words:

```
/start/,/stop/
```

- Print all lines whose first field differs from previous one:

```
$1 != prev {print; prev = $1}
```

awk

- Delimiter-separated fields:

  BEGIN {FS="c"}

- Example
  - `awk '{print $2, $1}' nums`
  -

```
1;1;1;1;1
2;2;2;2;2
3;4;5;6;7
4;9;9;0;1
5;5;6;7;2
6;1;1;1;6
7;9;8;0;0
8;8;1;1;2
9;2;5;4;3
```

Awk

Jimmy the Weasel
100 Pleasant Drive
San Francisco, CA 12345

Big Tony
200 Incognito Ave.
Suburbia, WA 67890

- Example: Addresses
- Put each address on same line
- Create file

```
BEGIN {
      FS="\n"
      RS=""
}
{
      print $1 ", " $2 ", " $3
}
```

- Run it:
  - `awk -f processAddr.awk addresses.txt`

1. Awk reads the input files one line at a time.
2. For each line, it matches with given pattern in the given order, if matches performs the corresponding action.
3. If no pattern matches, no action will be performed.
4. If the search pattern is not given, then Awk performs the given actions for each line of the input.
5. If the action is not given, print all that lines that matches with the given patterns which is the default action.
6. Empty braces with out any action does nothing. It wont perform default printing operation.
7. Each statement in Actions should be delimited by semicolon.

http://www.thegeekstuff.com/2010/01/awk-introduction-tutorial-7-awk-print-examples/

awk applications

- Unix file processing, e.g., as part of a pipe
- Avoid read loop in shells, Perl
- More intuitive syntax than shell scripts
- Best limited to line-at-a-time processing

References

- http://www.cs.arizona.edu/classes/cs352/spr13/NOTES/01%20-%20Basic%20Unix.ppt