Organization of a computer system

users

graphical user interface (GUI)

shell

operating system

hardware
(or software acting like hardware: “virtual machine”)

applications
Organization of a computer system

“The Internet”

user computer

network

server

lectura.cs.arizona.edu

graphical user interface (GUI)
shell
operating system
hardware (or software acting like hardware: “virtual machine”)
Easier to use; Not so easy to program with, automate

Organization of a computer system

Users

Applications

Interactive actions (click, drag, tap, …)

Graphical user interface (GUI)

System calls

Operating system

Hardware

(or software acting like hardware: “virtual machine”)
**Organization of a computer system**

Easier to program with and automate; Not so convenient to use (maybe)

- Users
- Applications
- Typed commands
- Graphical user interface (GUI)
- System calls
- Operating system
  - Hardware (or software acting like hardware: “virtual machine”)

Software layers:
- **Hardware**: The physical components of the computer system.
- **Operating System**: Manages the hardware and provides an interface for applications and users.
- **Applications**: Software programs that run on the operating system.
- **Graphical User Interface (GUI)**: The visual interface for applications.
- **Shell**: The command line interface for users to interact with the operating system.

Software architecture:
- Users interact with applications through typed commands or a GUI.
- Applications run on the operating system.
- The operating system communicates with the hardware through system calls.
Organization of a computer system

- hardware (or software acting like hardware: “virtual machine”)
- operating system
- shell
- graphical user interface (GUI)
- applications
- users

this class
Reading

• Chapter 1:
  – Upto “Background Jobs” (page 17)
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
Layers of a Unix system

- Users
- Applications
- Shell
- Unix operating system kernel
- Hardware
- System calls
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:

  \[ \text{command} \quad \text{options} \quad \text{arguments} \]

Examples:

<table>
<thead>
<tr>
<th>Command</th>
<th>Options</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>pwd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cd</td>
<td>-a -l</td>
<td>/home/debray</td>
</tr>
<tr>
<td>ls</td>
<td>-al</td>
<td>/usr/local</td>
</tr>
</tbody>
</table>

Options can (usually) be combined together: these are equivalent
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:

<table>
<thead>
<tr>
<th>Command</th>
<th>Options</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>pwd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cd</td>
<td></td>
<td>/home/debray</td>
</tr>
<tr>
<td>ls</td>
<td>-a -l</td>
<td>/usr/local</td>
</tr>
<tr>
<td>ls</td>
<td>-al</td>
<td></td>
</tr>
</tbody>
</table>

Not always required: may have default values

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

Defaults to current directory
Examples of Unix commands I

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

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

  *Examples:*

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cd /</code></td>
<td>move to the root of the file system</td>
</tr>
<tr>
<td><code>cd ~</code></td>
<td>move to one’s home directory</td>
</tr>
<tr>
<td></td>
<td>(also: just “cd” by itself)</td>
</tr>
<tr>
<td><code>cd /usr/local/src</code></td>
<td>move to <code>/usr/local/src</code></td>
</tr>
<tr>
<td><code>cd ../..</code></td>
<td>move up two levels</td>
</tr>
</tbody>
</table>
Examples of Unix commands II

• Command: `ls` — *lists the contents of a directory*
  – Examples:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ls</code></td>
<td>list the files in the current directory</td>
</tr>
<tr>
<td></td>
<td><em>won’t show files whose names start with ‘.’</em></td>
</tr>
<tr>
<td><code>ls /usr/bin</code></td>
<td>list the files in the directory <code>/usr/bin</code></td>
</tr>
<tr>
<td><code>ls -l</code></td>
<td>give a “long format” listing (provides additional info about files)</td>
</tr>
<tr>
<td><code>ls -a</code></td>
<td>list all files in the current directory, including those that start with ‘.’</td>
</tr>
<tr>
<td><code>ls -al /usr/local</code></td>
<td>give a “long format” listing of all the files (incl. those starting with ‘.’) in <code>/usr/local</code></td>
</tr>
</tbody>
</table>
Executing commands

Typing a command name at the **bash** prompt and pressing the **ENTER** key causes the command to be executed.

The command's output, if any, is displayed on the screen. Examples:

```bash
% hostname
lectura.cs.arizona.edu
% whoami
eanson
% true
% date
Sat Aug 15 18:54:39 MST 2015
% ps
    PID TTY         TIME CMD
 22758 pts/18   00:00:00 bash
 30245 pts/18   00:00:00 ps
```
Most commands accept one or more arguments:

```
% cal 9 2015
  September 2015
Su Mo Tu We Th Fr Sa
  1  2  3  4  5
  6  7  8  9 10 11 12
 13 14 15 16 17 18 19
 20 21 22 23 24 25 26
 27 28 29 30

% echo Hello, world!
Hello, world!

% factor 223092870
223092870: 2 3 5 7 11 13 17 19 23
```
Many commands accept *options* that adjust the behavior of the command.

Options almost always begin with a '-' (minus sign). Options are usually specified immediately following the command. For most programs the ordering of options is not significant but that is a convention, not a rule.

Examples:

```
% date
Thu Jan 13 02:19:20 MST 2005

% date -u
Thu Jan 13 09:19:22 UTC 2005

% wc Hello.java
  5     14     127 Hello.java

% wc -l -w Hello.java
  5     14 Hello.java
```

We can say that `wc -l -w Hello.java` has two options and one operand.
Whitespace is often significant in command lines. For example, the following commands are all invalid: (Try them!)

\% \texttt{date-u}

\% \texttt{wc -l-w Hello.java}

\% \texttt{wc -- notes Hello.java}
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)
    o 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 "/"
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 “/”
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
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

• Notational shorthand:
  – one’s own home directory:  ~
  – some other user joe’s home directory:  ~joe
Some commands for dealing with files

- `pwd`
  - *print the name of the current/working directory*

- `ls [file]`
  - list a directory contents

- `cd [dir]`
  - change the current/working directory

- `cp file_1 file_2`
  - copy `file_1` to `file_2`

- `vi [file]`
  - the vi editor

- `vimtutor`
  - a tutorial for using vi
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)
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`
  - `cmd₁ | cmd₂` redirect `cmd₁`’s stdout to `cmd₂`’s stdin
Combining commands

• The output of one command can be fed to another command as input.
  – Syntax:  \( \text{command}_1 \mid \text{command}_2 \)

Example:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ls</code></td>
<td>lists the files in a directory</td>
</tr>
<tr>
<td><code>more foo</code></td>
<td>shows the file <code>foo</code> one screenful at a time</td>
</tr>
<tr>
<td>`ls</td>
<td>more`</td>
</tr>
</tbody>
</table>

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

\texttt{man \ command}

“displays the on-line manual pages”

• Provides information about command options, arguments, return values, bugs, etc.
Example: “man ls”

<table>
<thead>
<tr>
<th>NAME</th>
<th>ls - list directory contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNOPHIS</td>
<td>ls [OPTION]... [FILE]...</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>List information about the files (the current directory by default). Sort entries alphabetically if none of <code>-crtwSUX</code> nor <code>-sort</code>. Mandatory arguments to long options are mandatory for short options too.</td>
</tr>
<tr>
<td>-a, --all</td>
<td>do not ignore entries starting with .</td>
</tr>
<tr>
<td>-A, --almost-all</td>
<td>do not list implied . and ..</td>
</tr>
<tr>
<td>--author</td>
<td>with -i, print the author of each file</td>
</tr>
<tr>
<td>-b, --escape</td>
<td>print octal escapes for nongraphic characters</td>
</tr>
<tr>
<td>--block-size=SIZE</td>
<td>use SIZE-byte blocks</td>
</tr>
<tr>
<td>-B, --ignore-backups</td>
<td>do not list implied entries ending with -</td>
</tr>
<tr>
<td>-c, --time</td>
<td>with -It: sort by, and show, ctime (time of last modification of file status information) with -I: show ctime and sort by name otherwise: sort by ctime</td>
</tr>
<tr>
<td>-c</td>
<td>list entries by columns</td>
</tr>
<tr>
<td>--color=[WHEN]</td>
<td>control whether color is used to distinguish file types. WHEN may be 'never', 'always', or 'auto'</td>
</tr>
</tbody>
</table>
Example: “man man”

NAME
man - format and display the on-line manual pages

SYNOPSIS

DESCRIPTION
man formats and displays the on-line manual pages. If you specify section, man only looks in that section of the manual. name is normally the name of the manual page, which is typically the name of a command, function, or file. However, if name contains a slash (/) then man interprets it as a file specification, so that you can do man ./foo.c or even man /cd/You/bar.1.gz.

See below for a description of where man looks for the manual page files.

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 Deansons

Distributions customize the manual section to their specifics, which often include additional sections.

OPTIONS
-1 config_file
   Specify the configuration file to use; the default is /etc/man.config. (see man.config(5).)
Example: “man man”

we can specify what kind of information we want
Some other useful commands

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

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

• **head** −n [file]
  • show the first $n$ lines of the input

• **tail** −n [file]
  • show the last $n$ lines of the input

• **cp** file$_1$ file$_2$
  • copy file$_1$ to file$_2$

• **mv** file$_1$ file$_2$
  • move file$_1$ to file$_2$
The diff command

• The diff command looks for differences in files
• You will probably want to know this command since it will be used in grading.
• Down the line you will be doing programming homework.
• An correct executable will be given.
• You'll want to use I/O redirection and the diff command to make sure the output of your code matches the output of the given program.
Odds and ends

• Here are some handy options for `ls`:
  • `-t` Sort by modification time instead of alphabetically.
  • `-h` Show sizes with human-readable units like K, M, and G.
  • `-r` Reverse the order of the sort.
  • `-S` Sort by file size
  • `-d` By default, when an argument is a directory, `ls` operates on the entries contained in that directory. `-d` says to operate on the directory itself. Try "ls -l ." and "ls -ld .".
  • `-R` Recursively list all the subdirectories.
• There are many more and you might want to look at the man page and play
Odds and ends

- Two handy options for `cp`:
  - `-R` Recursively copy an entire directory tree
  - `-p` Preserve file permissions, ownerships, and timestamps
Odds and Ends

• Many non-alphanumeric characters have special meaning to shells.

• Characters that have special meaning are often called *metacharacters*. Here are the *bash* metacharacters:

• ~ ` ! # $ & * ( ) \ | { } [] ; ' " < > ?
• Many non-alphanumeric characters have special meaning to shells.

• Characters that have special meaning are often called *metacharacters*. Here are the *bash* metacharacters:

```bash
~ ` ! # $ & * ( ) \ | { } [ ] ; ' " < > ?
```

• If an argument has metacharacters or whitespace we suppress their special meaning by enclosing the argument in quotes.
Odds and Ends

• Many non-alphanumeric characters have special meaning to shells.

• Characters that have special meaning are often called *metacharacters*. Here are the *bash* metacharacters:

  ~ ` ! # $ & * ( ) \ | { } [ ] ; ' " < > ?

• If an argument has metacharacters or whitespace we suppress their special meaning by enclosing the argument in quotes.

• An alternative to wrapping with quotes is to use a backslash to "escape" each metacharacter.
Pattern matching in the shell

• Some metacharacters are used as patterns in shell commands, e.g.:

\[
\begin{align*}
* & \text{ matches any string} \\
[ \ldots ] & \text{ matches any one of the characters within braces}
\end{align*}
\]

Example:

- `ls b*` list files that begin with `b` and end with `c`
- `ls a[xyz]*` list files starting with `a` followed by `x`, `y`, or `z`
- `ls *.pdf` list files ending with “.pdf”
bash supports simple command line recall and editing with the "arrow keys" but many control-key and escape sequences have meaning too. Here are a few:

\(^A/\ ^E\) Go to start/end of line.
\(^W\) Erase the last "word".
\(^U\) Erase whole line. (\(^C\) works, too.)
\(^R\) Do incremental search through previous commands.
\(\text{ESC}-\ f/\ b\) Go forwards/backwards a word. (Two keystrokes: \text{ESC}, then \text{f})
\(\text{ESC}-\ .\) Insert last word on from last command line. (Very handy!)

\textbf{bash} also does command and filename completion with \text{TAB}:

Hit \text{TAB} to complete to longest unique string.
If a "beep", hit \text{TAB} a second time to see alternatives.
Getting more information about files

- `ls -l`: provides additional info about files
Getting more information about files...

```
file name  last-modified time  size  group  owner  no. of hard links  access permissions  file type
---  -------------------  ------  -----  ----  ----------------  ----------------  ----
icon  2009-11-24 13:17  4096  icon  gmt  23  drwxrwsr-x  normal file
index.html  2009-12-22 16:41  3599  jharriso  jharriso  1  rw-r--r--  directory
intranet  2008-08-05 11:20  4096  officweb  gmt  5  drwxrwsr-x  normal file
lost+found  1996-06-07 09:32  4096  root  root  2  drwx------  symbolic link
```
# File access permissions

<table>
<thead>
<tr>
<th>Access</th>
<th>Owner</th>
<th>Group</th>
<th>Others</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>– – – –</td>
<td>root</td>
<td>root</td>
<td>– – – –</td>
<td>no permission</td>
</tr>
<tr>
<td>drwxrwsr-x</td>
<td>23 gmt</td>
<td>icon</td>
<td>4096</td>
<td>2009-11-24 13:17 icon</td>
</tr>
<tr>
<td>–rw-r-r--</td>
<td>1 jharriso jharriso</td>
<td>3599</td>
<td>2009-12-22 16:41 index.html</td>
<td></td>
</tr>
<tr>
<td>drwxrwsr-x</td>
<td>5 gmt officeweb</td>
<td>4096</td>
<td>2008-08-05 11:20 intranet</td>
<td></td>
</tr>
<tr>
<td>drwx------</td>
<td>2 root</td>
<td>root</td>
<td>4096</td>
<td>1996-06-07 09:32 lost+found</td>
</tr>
</tbody>
</table>

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

**Access Permissions**

- **access permissions for owner (u)**
- **access permissions for group (g)**
- **access permissions for others (o)**
Changing file access permissions

Command:

```
chmod who [what] file1 file2 ... file_n
```

∈ {r, w, x}

∈ {a, u, g, o}

Example:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chmod u-w foo</td>
<td>remove write permission for user on file foo</td>
</tr>
<tr>
<td>chmod g+rx bar</td>
<td>give read and execute permission to group for bar</td>
</tr>
<tr>
<td>chmod o-rwx *.doc</td>
<td>remove all access permissions for “other users” (i.e., not owner or group members) for *.doc files</td>
</tr>
<tr>
<td>chmod a+rw p*</td>
<td>give read and write permission to everyone for all files starting with p</td>
</tr>
</tbody>
</table>
Pattern matching: grep

GREP(1)

NAME
grep, egrep, fgrep - print lines matching a pattern

SYNOPSIS
grep [options] PATTERN [FILE...]
grep [options] [-e PATTERN | -f FILE] [FILE...]

DESCRIPTION
Grep searches the named input FILES (or standard input if no files are named, or the
file name - is given) for lines containing a match to the given PATTERN. By default,
grep prints the matching lines.

In addition, two variant programs egrep and fgrep are available. Egrep is the same as
grep -E. Fgrep is the same as grep -F.

OPTIONS
-A NUM, --after-context=NUM
   Print NUM lines of trailing context after matching lines. Places a line con-
taining — between contiguous groups of matches.

-a, --text
   Process a binary file as if it were text; this is equivalent to the --binary-
files=text option.

-B NUM, --before-context=NUM
   Print NUM lines of leading context before matching lines. Places a line con-
taining — between contiguous groups of matches.

-C NUM, --context=NUM
   Print NUM lines of output context. Places a line containing — between contigu-
Pattern matching: grep...

- print the current directory: `pwd`
- show the contents of this file: `cat GettysburgAddress`
- print out the lines that match "nation":
  ```
  grep 'nation' GettysburgAddress
  ```
Pattern matching: grep...

"print all lines in the input that match the string er"
pattern matching: grep...

- `grep -E 'er|re'`:
  - "print all lines in the input that match the string \texttt{er} or \texttt{re}\"

- `grep -E '^er|re'`:
  - "print all lines in the input that begin with the string \texttt{er} or \texttt{re}\"