System Programming in C

Reichman University
Efi Arazi School of Computer Science

Ilan Gronau
Sara H. Geizhals

Lecture overview

- Brief intro to course ←
- Course server
- File system basics
- Navigating file system (**`pwd`**, **`cd`**, **`ls`**)
- Manipulating files and directories (**`chmod`**, **`cp`**, **`mv`**, **`rm`**, **`mkdir`**, **`rmdir`**)

Course Overview

In this course we dive **deeper into programming principles**:

Program structure

- Data representation – type / scope / storage classes
- Memory organization of programs – stack vs. heap

Memory management

- Working with pointers
- Dynamic memory allocation
- Implementing objects using C modules (toward OOP…)

Course Overview

In this course we learn **new programming skills**:

Linux environment

- Working with UNIX-based OS (remote server and local VM)
- Using text as interface – Linux text parsing tools
- Scripting

Programming in C

- Type conversions & operators
- Working with arrays / pointers
- How to structure your program into files (modules)?

Course Overview

Where will I use these skills?

Linux and C programming provide an unbeatable combination for performance computing where we need:

- Speed
- Reliability (no crashes)

Very popular in:

- Systems-level programming
- High-performance scientific computing

For more on the joint history of C and Linux see:

- slides under “General resources” on the Piazza Resource page
- video guide on the course Panoptro folder

Course structure / syllabus

Linux (classes 1-4)

- The Linux file system, text parsing, shell scripting

Programming in C (classes 5-9)

- Data representation, functions and organization of stack
- Pointers, memory allocation and management

OOP in C (classes 10-11)

- Structures, modules
  (projects with multiple source files ← Lecture 12)

Course requirements

Grading policy

- Final Grade = 80% exam + 20% exercises
- Important: Only students who get an average passing grade on the HW exercises are eligible to take the exam!!!
- If you are in MILLUIM service and can't make the assignment deadline, contact us in advance and we'll set an alternative assignment schedule for you. Try to contact us as early as possible.

HW Exercises – philosophy and policy

- 5-6 programming exercises
- Each exercise covers a different topic in the course
- Individually solving the exercises is the best way to learn the material and reach the final exam prepared
- Every exercise counts into grade
  (see comment in previous slide about MILLUIM)
- All exercises count equally (3⅓ points from final grade)
- An unsubmitted exercise gets a grade of 0
- Better to submit a partial solution than skip an exercise

HW exercises – sharing ideas, not code

- You should work and submit HW exercises in pairs (working alone is okay, but not encouraged).
- Start by solving individually and then merge into one combined solution.
- You may discuss the exercise with other students, but make sure you are **not sharing code**
  - If you put your code in a shared space, you're sharing code.
  - If you send or receive a source file or copied text from a source file, you're sharing code.
  - If you are writing your code while having someone else's code in front of you, you're sharing code.

HW exercises – sharing ideas, not code

- You should work and submit HW exercises in pairs (working alone is okay, but not encouraged)
- Start by solving individually and then merge into one combined solution.
- You may discuss the exercise with other students, but make sure you are **not sharing code**
- The course graders periodically look for **suspiciously similar bits of code** in your solutions. Finding these is easier than you think!

HW exercises – working on the exercise

- Exercises are typically allocated 2 weeks (HW #1 allocated 1 week)
- We typically publish an exercise a few days before the deadline of the previous exercise. Use this time wisely.

  Example:
  - HW #1 will be published on Wed 27/3 and will be due Sun 7/4
  - HW #2 will be published on Wed 3/4 and will be due Sun 21/4
  - HW #3 will be published on Thurs 17/4 and will be due after Pesach . . .
- Exercises contain detailed instructions + guidelines for validation/checking.
- Make sure you save enough time for validation before submission.

HW exercises – submission

- Submission is done on the course server (`sysprog.runi.ac.il`)
- Detailed guide on checking your solution and submitting it in the **Homework submission guidelines** document on Piazza Resource page (see general Resources at the bottom of page).
- `check_ex` script for checking your solution before submission will give you instant feedback on errors.
- `submit_ex` script for submitting your solution.

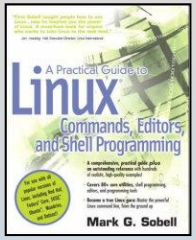
- Late submission:
  - Extension requests will be considered in special cases and only if submitted > 48 hours before deadline
  - If your extension request is not granted, you will be penalized 10 points for every 24 hours

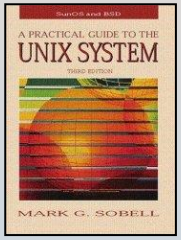
Other ways to practice material

- The course has a one-hour recitation (תרגולים) devoted to demonstrations and class exercises
- We conduct **class exercises** in many lectures. A laptop is not required but will be helpful.

Course books

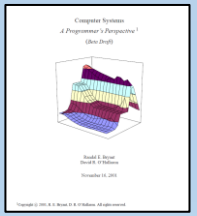
- Mark G. Sobell,
  A Practical Guide to the UNIX System (3rd Edition)
- Mark G. Sobell,
  A Practical Guide to the Linux Commands, Editors, and Shell Programming





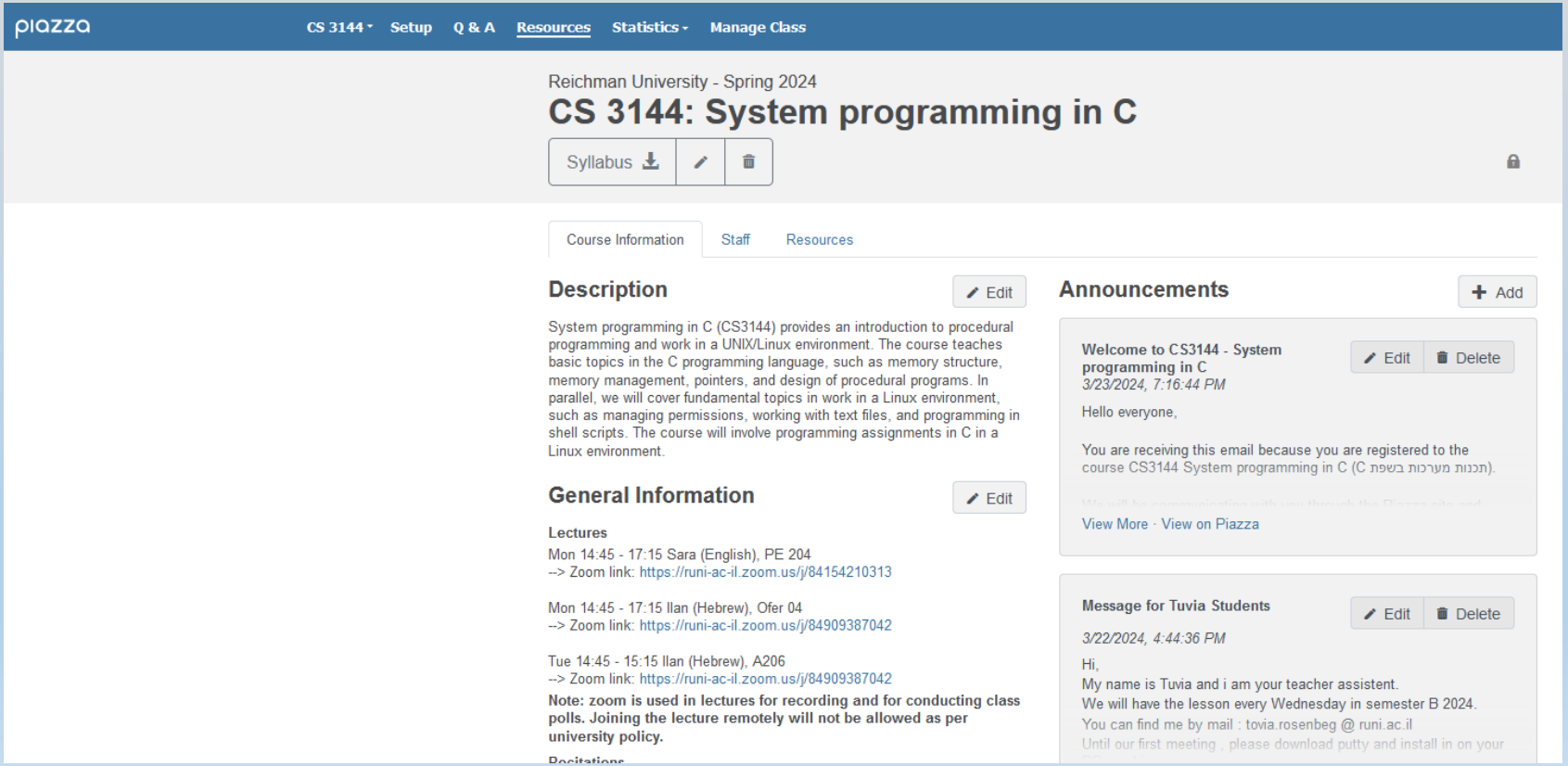
- Al Kelley and Ira Pohl,
  A Book on C: Programming in C (4th Edition)
- Brian Kernighan and Dennis Ritchie,
  The C Programming Language (2nd Edition)

- Randal E. Bryant, David R. O'Hallaron,
  Computer systems: a programmer's perspective



Course website

- Most correspondence will be done through Piazza
- If you didn't receive an invitation by email, then email one of the instructors



Contact us

- Posts through the Piazza website
- Email:
  - Sara H. Geizhals shgeizhals@gmail.com
  - Ilan Gronau ilan.gronau@runi.ac.il
  - Liam Tal liam.tal@post.runi.ac.il
  - Tuvia Rosenberg rtuvia197@gmail.com
- Office hours – locations and times updated on Piazza

Lecture overview

- Brief intro to course
- Course server ←
- File system basics
- Navigating file system (**`pwd`**, **`cd`**, **`ls`**)
- Manipulating files and directories (**`chmod`**, **`cp`**, **`mv`**, **`rm`**, **`mkdir`**, **`rmdir`**)

Accessing Linux

- Work in this course will be done on RUNI's Linux server:
  `sysprog.runi.ac.il`

- Ways of accessing the server:
  - From Windows:
    use remote client (such as PuTTY)
  - From Mac or Linux terminal:
    `ssh <username>@sysprog.runi.ac.il`

- If you wish to work locally and transfer files to the server, you can do so using a secure FTP client (such as PuTTY SFTP or WinSCP)

- For detailed instructions on how to access server, see:
  - detailed log in guide on the Piazza Resource page
  - video guide on the course Panoptro folder

Local Linux installation

- There are many Linux variants (flavors)
  Recommended: Ubuntu, Fedora, Mint
- Each can be used in two ways
  - Full install – not simple, and somewhat risky
  - Live media – good as a demo, but has many disadvantages for real work
- It is also possible to create a Linux virtual machine that will operate within another OS
  (using VirtualBox or VMware)
- On Windows 10, it is also possible to install a Linux shell emulator

Poll 1

Poll 1

How many assignments are in the course?

- 4
- 5-6
- 7
- 8

Submission of the HW assignments is done

- Via Piazza
- Via the course server
- Via email
- On paper

If I don’t submit an assignment …

- No problem - there's the (n-1) rule.
- I will get a grade of 0 for that assignment.
- I will automatically fail the course.

If I don't find a partner for a HW assignment …

- I need to contact the instructors to ask permission to submit alone.
- I should ask around (on Piazza, WhatsApp, etc.) to find a partner.
- I will automatically fail the course.

Lecture overview

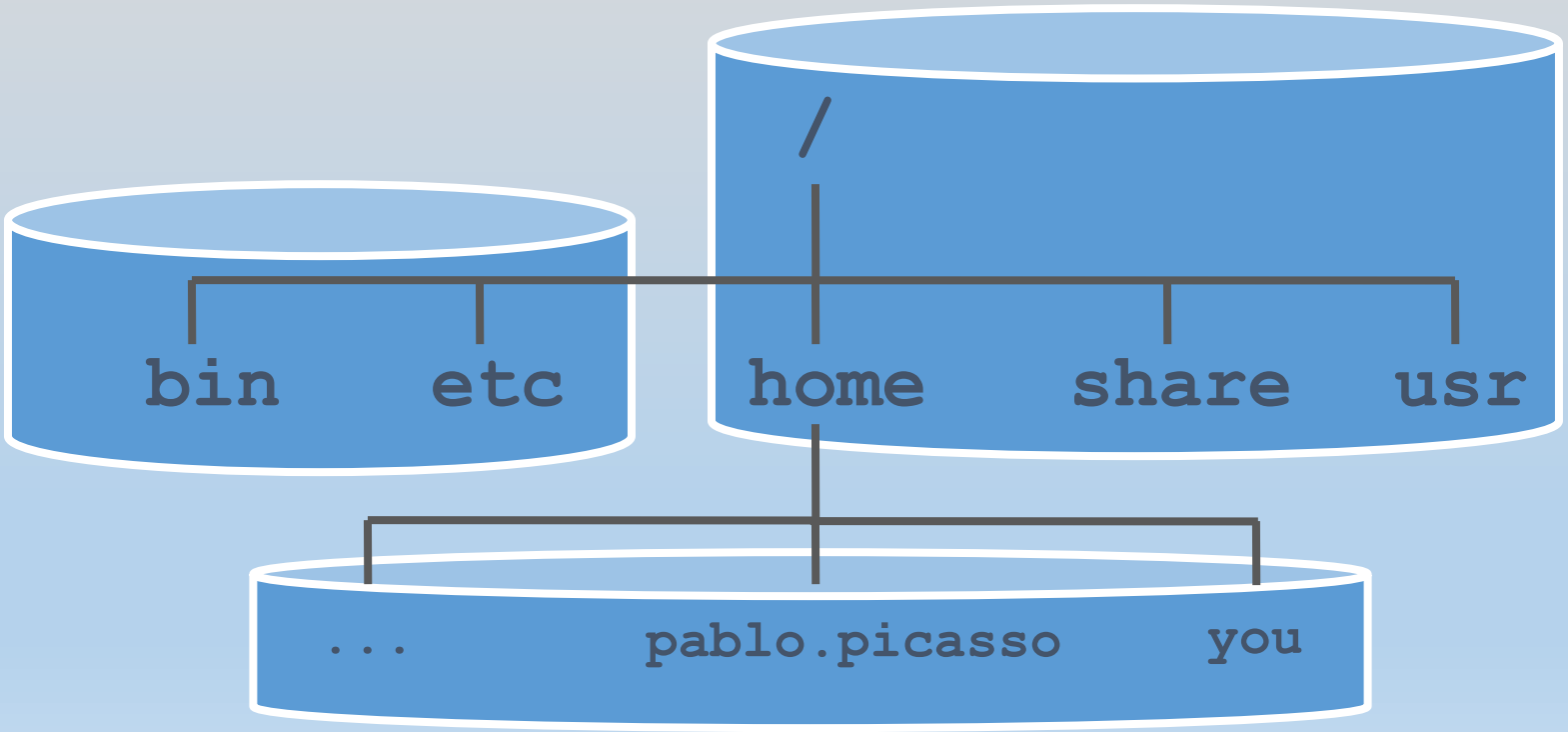
- Brief intro to course
- Course server
- File system basics ←
- Navigating file system (**`pwd`**, **`cd`**, **`ls`**)
- Manipulating files and directories (**`chmod`**, **`cp`**, **`mv`**, **`rm`**, **`mkdir`**, **`rmdir`**)

Linux/UNIX file system

- Files in the Linux operating system are arranged in a hierarchical structure
- Path components are separated by a forward slash – '/' (not by a backslash)
- At the top there is the root directory: '/'
  - [Directory = folder]
- Under it are various system directories
- Each user has his own hierarchy of files

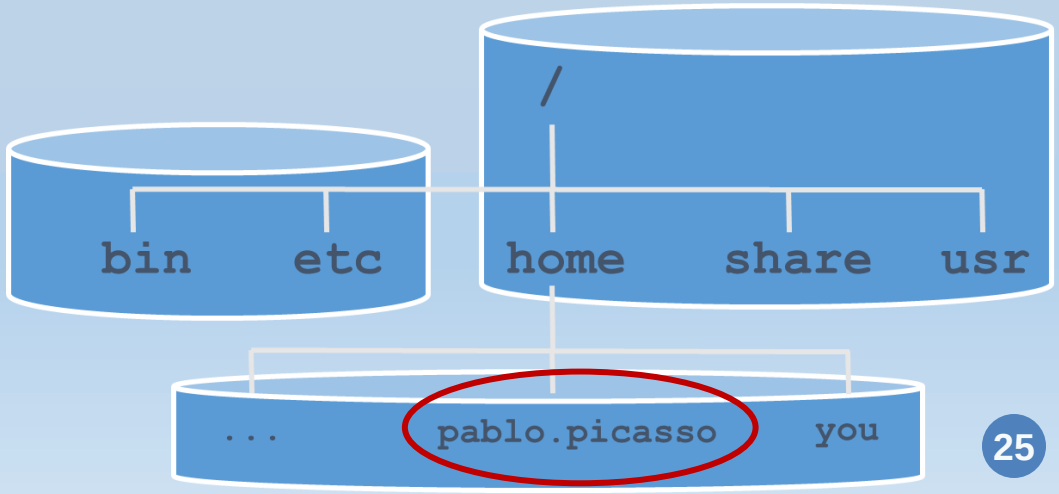
File system vs. physical disks

- The file hierarchy can be implemented using a number of physical disks



User home directory

- The home directories of users normally reside under `/home` (e.g., `/home/pablo.picasso`)
- Assuming you are Pablo Picasso – for your own homedir, use ~
  '~' <==> '`/home/pablo.picasso`'

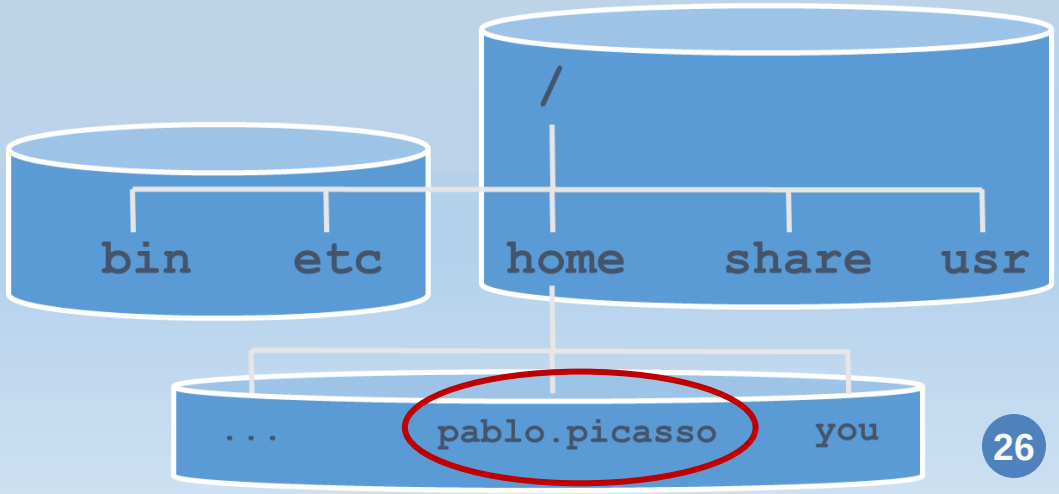


Absolute vs. relative paths

Any location in the filesystem can be specified using

- Absolute path from root '/'
- Relative path from current directory

If you are Pablo Picasso – the absolute path of your homedir is `/home/pablo.picasso`



Absolute vs. relative paths

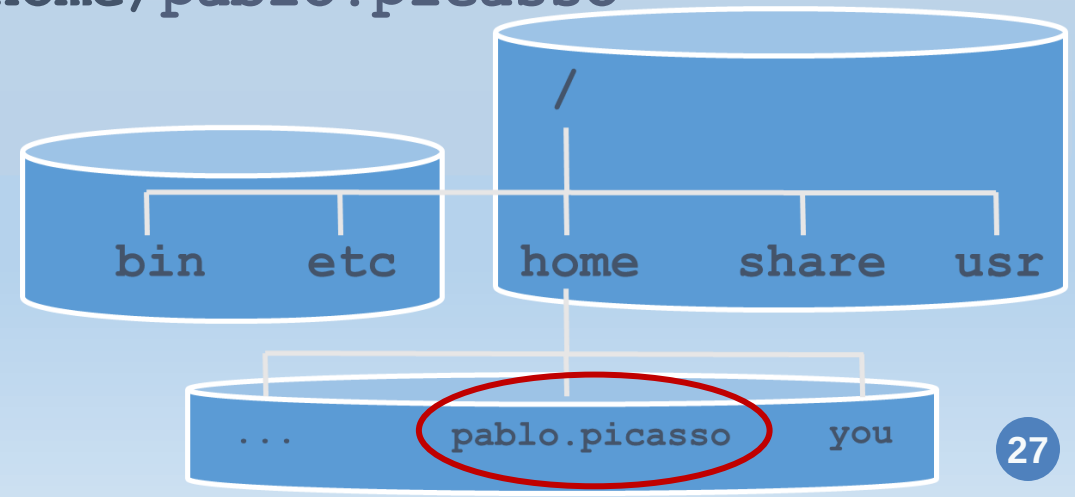
Any location in the filesystem can be specified using

- Absolute path from root '/'
- Relative path from current directory

If you are Pablo Picasso – the relative path of your homedir is

- Relative to `/home` ➔ `pablo.picasso`
- Relative to `/home/pablo.picasso` ➔ `.`
- Relative to `/share` ➔ `../home/pablo.picasso`

. (current dir)
.. (parent dir)



Conventions for directory names

- There are **no strict directory name conventions** in Linux
- Dir names can contain almost any sequence characters, but **we will typically restrict names to letters (upper or lower case), numbers, and a few symbols (‘_’ ‘.’ ‘–’)**
  - Strict rule: don’t use #, <, ?, =, +, \ etc.
- Dir names are **case sensitive**

Conventions for file names

- There are **no strict filename conventions** in Linux
- File names can contain almost any sequence characters, but **we will typically restrict names to letters (upper or lower case), numbers, and a few symbols (‘_’ ‘.’ ‘–’)**
  - Strict rule: don’t use #, <, ?, =, +, \ etc.
- File names are **case sensitive**
- The file extension is the string after the last ‘.’ in the name and it typically represents the file’s function (text, C code, Bash code, etc.)
- File extensions are **non-committing** and files don’t have to have extensions. Changing a file’s extension doesn’t do anything to contents of the file!

Lecture overview

- Brief intro to course
- Course server
- File system basics
- Navigating file system (**`pwd`**, **`cd`**, **`ls`**) ←
- Manipulating files and directories (**`chmod`**, **`cp`**, **`mv`**, **`rm`**, **`mkdir`**, **`rmdir`**)

Changing / displaying the current directory

- The `pwd` command displays the absolute path of the current directory

```
pwd
```

```
/home/pablo.picasso/exercises
```

- The `cd` command changes the current directory. Its general form is:

```
cd [directory-name]
```

- With no parameters, `cd` changes the current directory to your home directory

`ls` – Listing directory contents

The `ls` command lists the contents of some directory.

Examples:

```
ls
```

- Lists content of current directory

```
ls bin
```

- List the contents of directory **`bin`** (relative path)

```
ls a.txt
```

- If there exists a file called **`a.txt`** (relative path), then list it

```
ls a.txt bin
```

- Combination of the above

Filename expansions using wildcards

Use symbols to define a collection of files or directories or a collection of strings.

- `'*'` replaces any sequence of characters (also empty)
- `'?'` replaces a single character
- `[ ]` replaces a specific collection or range of characters
  (e.g., `[rwx]` – `'r'`,`'w'` or `'x'`,
  `[r-w]` – `'r'` to `'w'`, `[a-z]` – `'a'` to `'z'`)
- `[^ ]` replaces any character except those in the specified set
  (e.g., `[^rwx]` – any character other than `'r'`,`'w'` or `'x'`)
- `{ , , }` replaces a collection of strings
  (e.g., `{test,text,rest}` – matches either `'test'`,`'text'` or `'rest'`)

Filename expansions using wildcards

Examples:

```
ls
```

```
boxes.c  DOC.txt   parse_tools.c  shape.c  tools.c
boxes.h  HELP.txt  parse_tools.h  shape.h  tools.h
```

```
ls *.c
```

```
boxes.c  parse_tools.c  shape.c  tools.c
```

- files (and directories) whose name ends with `'.c'`

```
ls *tools*
```

```
parse_tools.c parse_tools.h tools.c tools.h
```

- files (and directories) whose name contains `'tools'`

Filename expansions using wildcards

Examples:

```
ls
```

```
boxes.c  DOC.txt   parse_tools.c  shape.c  tools.c
boxes.h  HELP.txt  parse_tools.h  shape.h  tools.h
```

```
ls *tools.?
```

```
ls *tools.[ch]
```

```
parse_tools.c parse_tools.h tools.c tools.h
```

- files (and directories) whose name has `'tools'`, which is preceded by anything but is followed by just a period and one additional character
- files (and directories) whose name has `'tools'`, which is preceded by anything but is followed by `.c` or `.h`
- (in this case, same result)

Filename expansions using wildcards

Examples:

```
ls
```

```
boxes.c  DOC.txt   parse_tools.c  shape.c  tools.c
boxes.h  HELP.txt  parse_tools.h  shape.h  tools.h
```

```
ls {tools,shape}.?
```

```
shape.c shape.h tools.c tools.h
```

- files (and directories) whose name begins with either `'tools'` or `'shape'` and is followed by just a period and one additional character

Wildcard – important notes

- When a wildcard expression is used in the command line it is automatically converted by the Shell to the list of matching file / directory names. Thus, wildcards can be used in the context of any Linux command accepting file / directory names
- Wildcard expressions are similar to regular expressions, but they are not the same thing (more on that to come…)
- If you want to use a wildcard character (for instance `'?'`) in a string:
  - precede it with a backslash (e.g., `'\?'`)
  - or wrap it with double quotes (e.g., `'"?"'`)

Command line options

- Linux commands typically have optional flags they can take as input.
- Flags are typically indicated by a dash (`'-'`) followed by some letter, or in long form – double dash (`'--'`) followed by a string
- The most common options for `ls` are:
  - `-a` – all (also shows **hidden files**)
  - `-F` – include special char to indicate file types
  - `-l` – long format (include file size, times, permissions, etc.)

(see Linux Commands Guide)

- Options can be given separately

```
ls -a
```

```
ls -l
```

or together

```
ls -al
```

`ls` – Examples

```
ls
```

```
backup.tar.gz   create_backup   phone_list   temp
bin             my_documents    solutions
```

```
ls -F
```

```
backup.tar.gz   create_backup*   phone_list   temp/
bin/            my_documents/    solutions/
```

- File types:
  - Executable (suffix `*`)
  - Directory (suffix `/`)
  - Other (no suffix)
- File types are shown when using the `-F` option in `ls`

`ls` – Examples

`ls`

```
backup.tar.gz   create_backup   phone_list   temp
bin             my_documents    solutions
```

`ls -a`

```
.       .history        bin             phone_list
..      .bashrc         create_backup   solutions
.exrc   backup.tar.gz   my_documents    temp
```

- Hidden files:
  - Files and directories whose name starts with a dot (`'.'`) are not listed in a standard `ls` application.
  - Hidden files can be used for anything, but they are typically used for setup properties for other program:
    `bash` (`.bashrc`), `vi` (`.exrc`), `ssh` (`.ssh` directory)
  - Hidden files are listed when using the `-a` option in `ls`

Poll 2

Poll 2

Mark all the absolute paths

- /home/Donald.trump
- exercises/ex1
- ~/exercises/ex1/my_commands.txt
- ../classes/class01/myFile.txt

Which of these are valid names for text files?

- myFile.txt
- my.file
- myFile.exe
- myFile.exe1
- myFile{3}
- dir

Which of these are valid names for directories?

- dir
- my.dir
- file3
- file3.txt

`ls` – Examples

```
ls
```

```
backup.tar.gz   create_backup   phone_list   temp
bin             my_documents    solutions
```

```
ls -l
```

```
-rw-------  1 pablo.picasso  students   317 Oct 20 04:35 backup.tar.gz
drwx------  2 pablo.picasso  students  4096 Oct 19 04:31 bin
-rwxr-----  1 pablo.picasso  students    35 Oct 19 13:46 create_backup
drwx------  2 pablo.picasso  students  4096 Oct 19 04:30 my_documents
-rw-r--r--  1 pablo.picasso  students    17 Oct 19 04:31 phone_list
drwx------  3 pablo.picasso  students  4096 Oct 19 04:32 solutions
-rwxrw-r--  2 pablo.picasso  students  4096 Oct 19 04:30 temp
```

- The `-l` option is used for printing a detailed account for each file

File/dir permissions

The permissions have three types:

	File	Directory
Read (`r`)	file contents can be read and displayed	dir contents can be listed
Write (`w`)	file contents can be changed + file can be moved or deleted	dir attributes can be modified, e.g., can create, rename, or delete files within dir
Execute (`x`)	file can be executed as a program	dir can be entered into

Permissions

- Permissions are granted in three layers:
  - User/Owner – the user who created the file
  - Group – users within a specified group containing the owner
  - Others – all users in the system (other than user/owner and those in the group)

Permissions – examples

	Permissions		User	Group	Size	Date/Time	Name
1)	-rw-------	1	demo	students	317	Oct 19 04:35	backup.tar.gz
2)	drwx------	2	demo	students	4096	Oct 19 04:31	bin
3)	-rwxr-----	1	demo	students	35	Oct 19 13:46	create_backup
4)	drwxr-----	2	demo	students	4096	Oct 19 04:30	my_documents
5)	-rw-r--r--	1	demo	students	17	Oct 19 04:31	phone_list
6)	drwxr-x---	3	demo	students	4096	Oct 19 04:32	solutions
7)	-rwxrw-r--	2	demo	students	4096	Oct 19 04:30	temp

1) File can be read/modified only by owner
2) Directory can be read/modified/entered by owner
3) File can be read/modified/executed by owner + read by group (those in the group "students")
4) Directory has full access by owner + contents can only be read by group
5) File can be read by anyone + modified only by owner
6) Directory has full access by owner + it can be read and entered by group
7) File can be read by anyone + modified by owner and group + executed by owner

Lecture overview

- Brief intro to course
- Course server
- File system basics
- Navigating file system (**`pwd, cd, ls`**)
- Manipulating files and directories (**`chmod, cp, mv, rm, mkdir, rmdir`**) ←

`chmod` – Changing file permissions

- The `chmod` command changes the permissions associated with a file or directory

```
chmod [options] mode file...
```

- The format of the mode is: `[ugoa][+-][rwx]`
  - `ugoa` user (owner), group, other, or all
  - `+-` add or remove permission
  - `rwx` read, write or execute

chmod – Examples

```
chmod a-w phones
```

- Remove write permissions from all users

```
chmod +w phones
```

- Add write permissions to owner

```
chmod u+x my_script addresses
```

- Make a file executable to user

```
chmod ug+r-w my_script
```

- Make a file read only to user and group

```
chmod g+w my_script
```

- Gives write permissions to group, but not to user!

Note: make sure your permissions make sense

File permissions

The permission setting of a given file or directory can be represented by a binary vector of length 9

```
-rw-------  1 demo  students   317 Oct 19 04:35 backup.tar.gz
 110000000

drwxr-x---  3 demo  students  4096 Oct 19 04:32 solutions
 111101000
```

- We can directly specify the permission of a given file or directory by specifying this binary vector using “octal digits” (base 8)
- Each octal digit (0-7) represents 3 binary flags for the r/w/x/ permissions of owner, group, or others.

Example:

```
chmod 644 phones
```

Owner Group Other

```
rw-r--r--
110100100
 6   4   4
```

File permissions – examples

```
-rw-------  . . . backup.tar.gz
drwx------  . . . bin
-rwxr-----  . . . create_backup
drwxr-----  . . . my_documents
-rw-r--r--  . . . phone_list
drwxr-x---  . . . solutions
-rwxrw-r--  . . . temp
```

binary	octal
110 000 000	600
111 000 000	700
111 100 000	740
111 100 000	740
110 100 100	644
111 101 000	750
111 110 100	764

chmod – common octal modes

Mode	Meaning
777	Owner, group and public can read, write and execute file
755	Owner can read, write and execute file; group and public can read and execute file
644	Owner can read and write file; group and public can read file
700	Owner can read, write and execute file; group and public have no access to the file

`chmod` – common command line options

- `-R`
  operate recursively. Will set the permissions of all files and directories within the tree rooted at each specified directory
  - Note: -R and not –r, since –r might be confused with removing reading permission
- `-reference=[FILE]`
  Copy the permissions of a given reference file to another file

chmod – additional notes

```
chmod [options] mode file...
```

Two modes of operations:

- Alphabetical (relative) permission:
- Octal (absolute) permission:

```
Mode = [ugoa] [+-] [rwx]
```

```
Mode = [0-7] [0-7] [0-7]
```

Sensible application:

- You can assign a file with any of the 2^9=512 possible priority modes, but there are a few rules of thumb you should follow:
  - Don't remove the owner's read access
  - Maintain hierarchical priorities – owner ≥ group ≥ others
  - If you give read access ('r') to a directory, then also give executable ('x') permissions

File manipulation commands

`cp source_path dest_path`	**copy file(s)**
`mv source_path dest_path`	**move file(s)** [copies and deletes source]
`rm path`	**remove/delete file(s)**
`mkdir path`	**create new directory(ies)**
`rmdir path`	**delete directory(ies)** [only if empty !!]

- All path arguments above can be specified as **relative** or **absolute** paths
- `source_path` and `path` can contain **one or more paths**

Examples:

```
==> mkdir myDir yourDir
```

creates two sub-directories `myDir yourDir` under current directory

```
==> rm file1 file2
```

deletes files `file1 file2` from current directory

Copying directories

```
cp source_path dest_path
```

- If you wish to copy entire directories, specify the directory paths in `source_path`, and use **recursive mode** (option `-r` or `-R`)
- If `source_path` contains directory paths and recursive mode is not used, the operation will fail.

Example 1:

```
==> cp   hw.c      classes
```

Attempts to copy `hw.c` (source) from current dir to subdir `classes` (dest). Will work as `hw.c` is a file in the current dir and `classes` is a subdir of the current dir.

Copying directories

```
cp source_path dest_path
```

- If you wish to copy entire directories, specify the directory paths in `source_path`, and use **recursive mode** (option `-r` or `-R`)
- If `source_path` contains directory paths and recursive mode is not used, the operation will fail.

Example 2:

```
==> cp  /share/ex_data  .
```

Attempts to copy `ex_data` (source) from `/share` to current dir (dest).
Will fail because `ex_data` is a directory and not a file. To fix this we apply `-r`

```
==> cp -r /share/ex_data  .
```

Copying directories

```
cp source_path dest_path
```

- If you wish to copy entire directories, specify the directory paths in `source_path`, and use **recursive mode** (option `-r` or `-R`)
- If `source_path` contains directory paths and recursive mode is not used, the operation will fail.

Example 3:

```
==> cp  -r /share/ex_data file.txt  ~/exercises/ex1/
```

Attempts to copy (directory) `ex_data` from `/share` (source1) and (file) `file.txt` from current dir (source2) to directory `~/exercises/ex1/`. Will succeed if directory `~/exercises/ex1/`exists but fail otherwise.

Moving or renaming files/directories

`mv source_path dest_path`

- Command `mv` simply **copies and removes original**
- No need for recursive mode (`-r` or `-R`) when moving directories
- File/directory can be **renamed** by using the same directory and different file name in `source_path` and `dest_path`.

Example 1:

```
==> mv  ~/prog.txt  ~/prog.c
```

Rename file `prog.txt` from your home directory to `prog.c`

Example 2:

```
==> mv  ex1 ~
```

Move directory `ex1` from current dir to your homedir (with its contents)

Two modes for `cp` and `mv`

```
cp source_path dest_path
mv source_path dest_path
```

- **Mode I:** If `dest_path` is an **existing directory** then all files and dirs in `source_path` are copied/moved into directory `dest_path` with their original names.

  Note: `source_path` may be a list of paths

- **Mode II:** If `dest_path` is **not an existing directory** then `source_path` is copied/moved into the parent directory of `dest_path` with its name changed.

  Note: `source_path` must be a single path, and `dest_path` must be a path **under** an existing directory

Two modes for `cp` and `mv`

```
cp source_path dest_path
mv source_path dest_path
```

Examples:

```
==> cp   hw.c  ~/classes/class01
==> cp -r  ~/exercises  hw.c  ~/classes/class01
```

- If the dest path (`~/classes/class01`) is a path to an existing directory, then both operations are valid and will copy the source item(s) to the dest dir (`~/classes/class01`) with their original names.

Two modes for `cp` and `mv`

```
cp source_path dest_path
mv source_path dest_path
```

Examples:

```
==> cp   hw.c  ~/classes/class01
==> cp -r  ~/exercises  hw.c  ~/classes/class01
```

- If the dest path (`~/classes/class01`) does not exist, then the second operation will result in an error.
- However, if the path to the parent directory of the dest path (`~/classes`) does exist, then the first operation will copy the single source item (`hw.c`) to the parent dir (`~/classes`) with the new name `class01`.
- If the path to the parent directory of the dest path (`~/classes`) doesn't exist, then the first operation will also result in an error.

Delete files and directories

```
rm path
rmdir path
```

- Command `rm` is typically used to delete files.
- Command `rmdir` can delete a directory only if it is empty.
- A non-empty directory can be deleted by using `rm` in recursive mode (option `-r` or `-R`)

Caution caution caution!!!

- Linux does not have an ‘undo’ option for filesystem commands.
  Once you delete a file, it is gone!
- All operations are subject to the appropriate file permissions.
- Remove write permissions for files you wish to protect from accidental deletion.
- Use interactive mode (`-i`) to be prompted for each file that is overwritten or removed
- Use forced mode (`-f`) to remove without prompting
  (for write-protected files or non-existing files)

Linux command line help

- You may get basic help on usage of a Linux command by using the option `--help`

```
ls --help
```

```
Usage: ls [OPTION]... [FILE]...
List information about the FILEs (the current directory by
default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is
specified.

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 ..
...
```

- Square brackets `'[]'` in usage passage mean that an option or input argument is **optional** (mandatory arguments appear without brackets).

Linux command line help

- You can get detailed help on a Linux command by examining its **man** page (for manual).

```
man ls
```

```
LS(1)                    User Commands                    LS(1)

NAME
       ls - list directory contents

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

DESCRIPTION
       List
...
```

- The **man** pages of many Linux commands are also available on the web. See, e.g., http://linux.die.net/man/1/

Linux command line help

- The `man` pages of a given command can be navigated through using the following commands (from the `vi` editor):

Key	Meaning
`f, <Space>`	Move one page forwards
`b`	Move one page backwards
`<Enter>`	Move one line down
`/pattern`	Search for pattern
`n`	Repeat previous search
`q`	Quit viewing man page

Linux command line help

- A summary of common command options (not all of them) is given in the Linux Commands guide on the Piazza resource page.

Basic input / output

- The `echo` command can be used to display text

```
echo hello world
```

```
hello world
```

- Output can be redirected into a file using `>` and `>>`

```
echo hello world > file.txt
echo hello again >> file.txt
```

`file.txt` will contain the following:

```
hello world
hello again
```

- Redirection variants:
  - `>` – directs output to a new file (overwrites if file exists)
  - `>>` – appends output to an existing file (creates file if does not exist)

Displaying file contents

- `cat file` – displays the contents of a (text) file
- `more file` – displays the contents of a (text) file up to what can be viewed in one screen – can be scrolled down
- `less file` – similar to `more`, but you can scroll up and down and search (as you do with `man` pages)

More next week …

Text editing in Linux

- There are several text-based text editors for Linux (`pico`, `pine`, `vi`, `emacs` …)
- In this course we use `pico` and `vi`.
- Both editors are keyboard-based.
- `pico` is a bit more intuitive for simple use.
- `vi` is more powerful, but more difficult to get used to.
- We provide starter guides for both editors on the Piazza resource page.

At home…

- If you haven't received a welcome message from Piazza, email us to add you to the list.
- If you haven't done so already - log on to server and change your password!
- 1st homework assignment will be published on Wednesday (due Sunday, April 7 @ 21:00).
- Read HW Guidelines document on Piazza.
- Review relevant guides on Piazza:
  - Linux server (for logging in and changing password)
  - History of Linux/C
  - Linux commands (some of them; the rest TBD next lecture)
  - `pico` text editor