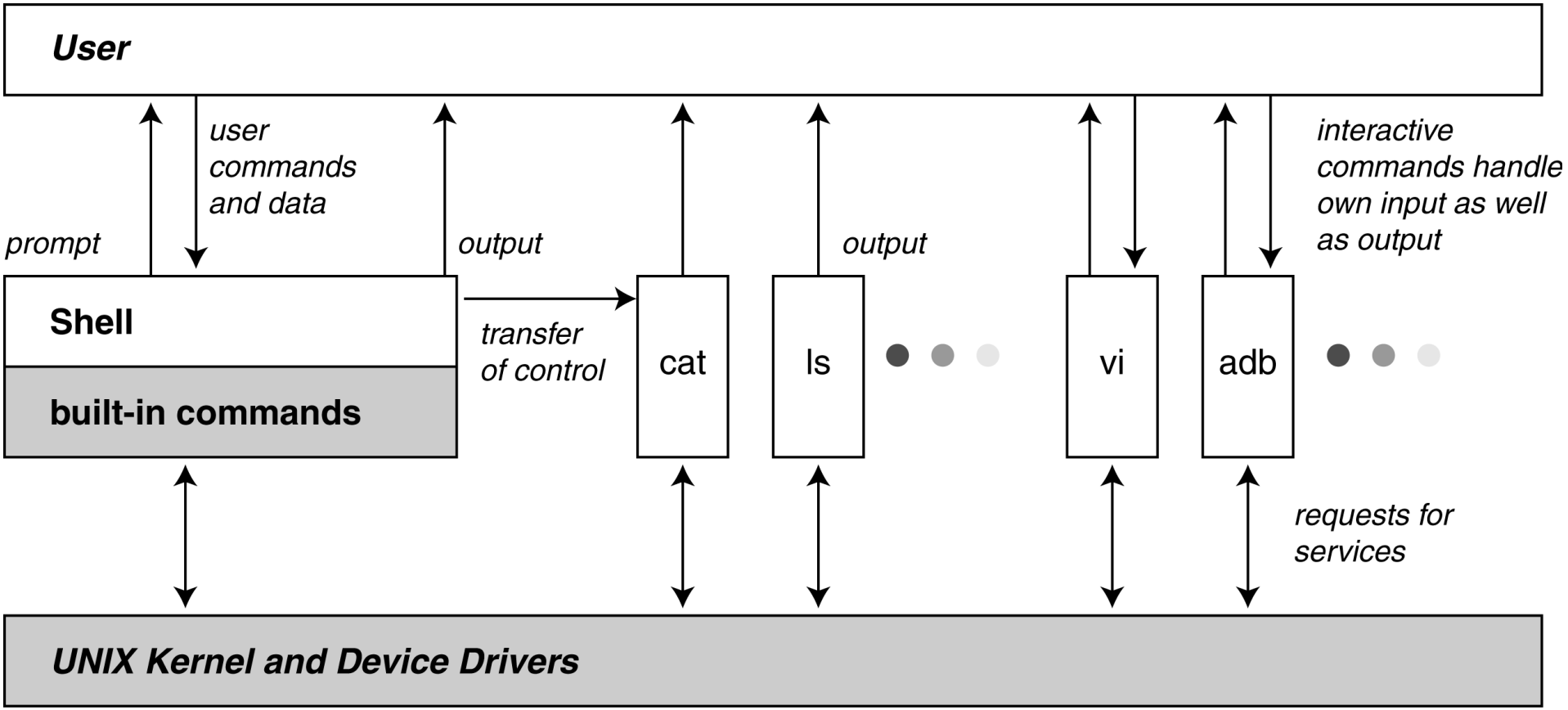
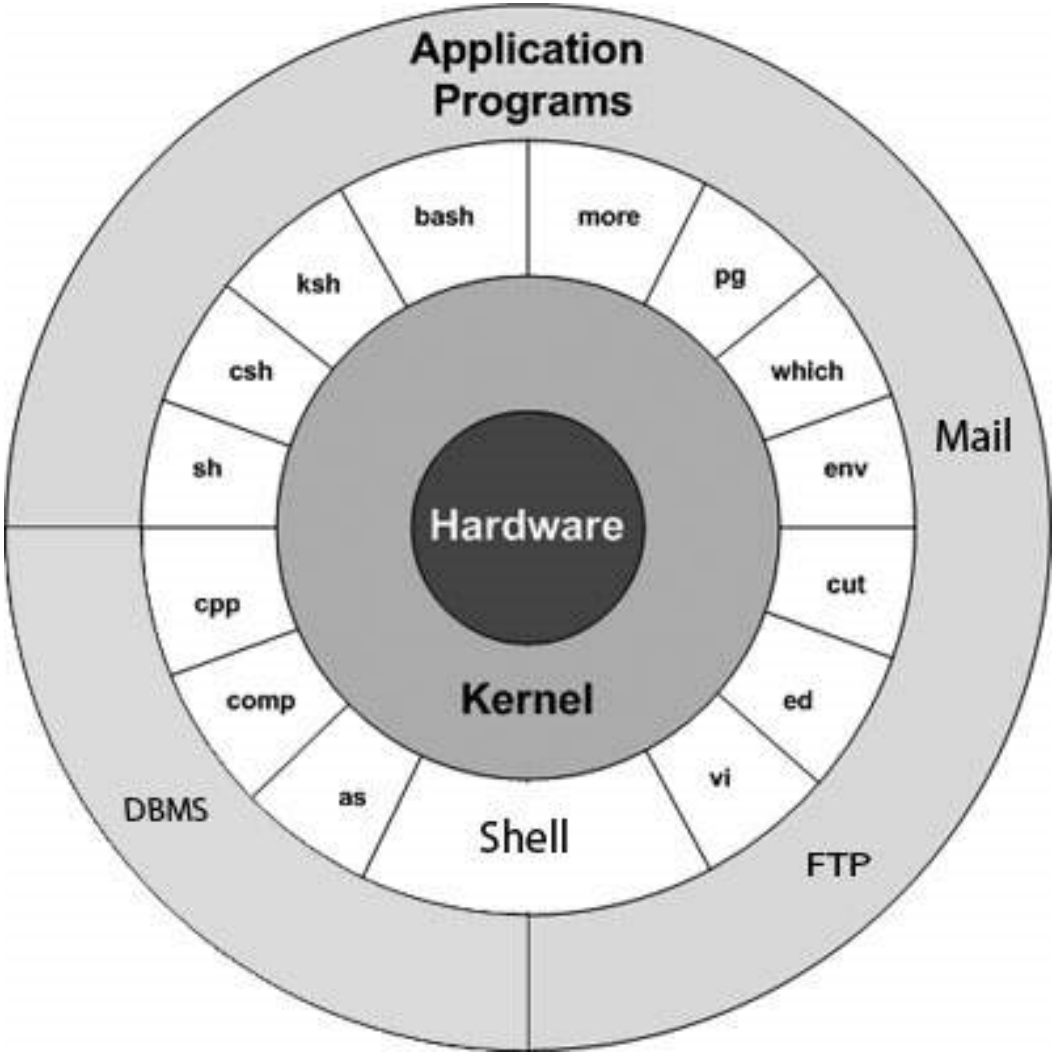
# Introduction to Unix

- System architecture
- File system
- File attributes (permissions)
- The shell
- Basic command syntax
- Command types
- Getting help on commands

# UNIX model of user interaction



# UNIX model


Application Programs
bash
more
pg
which
env
cut
ed
vi
Shell
as
comp
cpp
sh
csh
ksh
Hardware
Kernel
Mail
FTP
DBMS

# shell

- responsible for communication between the user and kernel
- “shell” refers to its position in some diagrams of UNIX structure
  - These diagrams use concentric rings to show layering
- reads and interprets user commands at the console (or from within a “shell script”)
- implements job control
- many shells have been developed:
  - `sh, csh, ksh, tcsh, zsh, bash …`
  - in this course we use the `bash` shell
  - `bash` extends `sh`, the Bourne shell

# kernel

- the **kernel** is the **protected core** of the **OS**
- the kernel is itself a large and complex program
- clear demarcation between the “kernel” and a “user”
  - to access a computer’s hardware (via the OS), user's commands must go through kernel
  - that is, “user” must request the kernel to perform work on behalf of “user”
  - user/OS interaction mediated by a command shell (e.g., **bash**), or the system library (compiled application)
- main responsibilities
  - memory allocation
  - **file system**
  - loads and executes programs (assumes a process model)
  - communication with devices (input, output)
  - bootstraps the system

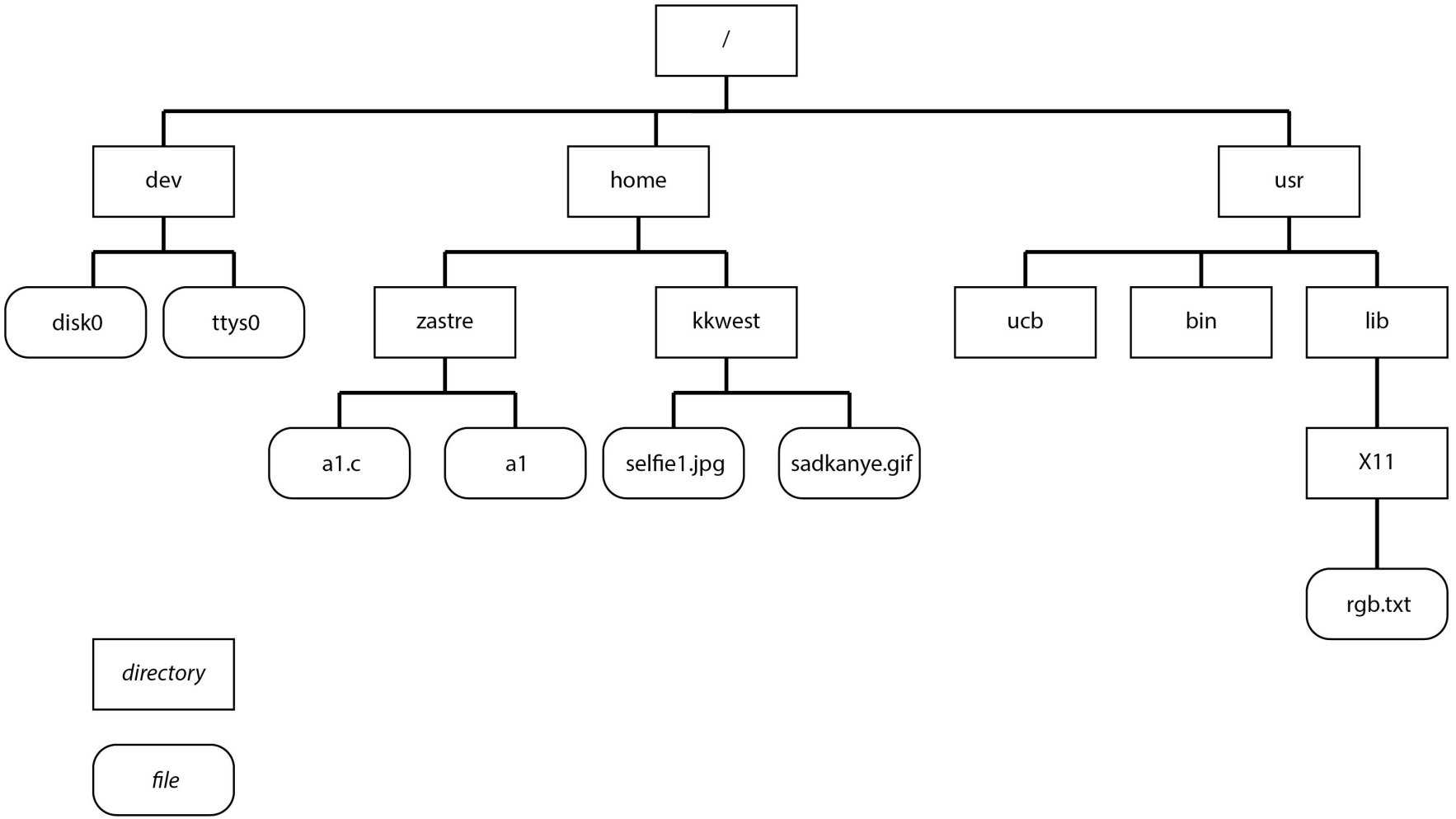
# UNIX filesystem

- "file", "filesystem": **are key abstractions** of the UNIX computing model
- practically anything can be abstracted as a file (devices, programs, data, memory, IPC, etc.)
- mainly responsible for mapping blocks of data within **physical** storage devices (hard drive, flash memory) onto **logical** blocks that users can manipulate
  - maps filenames to block numbers
  - handles block allocations; chains units together
  - provides methods to access data
- facilitates the "multiuser" view of the OS

# UNIX filesystem

- arranged as a hierarchy (tree-like structure)
  - organized as a collection of directories; think of a directory as a folder
  - forward slash "/" is used to separate directory and file components (cf., Windows uses "\")
- the root of the filesystem is represented by the **root-directory**, which we denote by a single "/"

# Part of a (hypothetical) UNIX filesystem tree



# Some properties of directories

- directories are actually “ordinary” files
- information contained in a directory file simply has a special format
- every directory contains two special directory entries
  - ".." refers to parent directory in hierarchy
  - "." refers to the current directory (itself)
- '~' is used to denote a **home directory**
  - % cd /home/*user* ≈ cd ~*user*
    % cd ≈ cd ~

# Directory commands (ignore %)

- Example:

  % cd /home

- listing directories

  % ls
  zastre keyboardcat
  % ls keyboardcat
  hi-rez.mp4 tinder-stuff.txt

- relative pathnames

  % cd /home
  % open keyboardcat/hi-rez.mp4
  % open ./keyboardcat/hi-rez.mp4
  % open ./keyboardcat/../keyboardcat/hi-rez.mp4

# “working” vs. “home” directory

- “Working” directory is the directory the shell determines you are “in” at any point in time.
  - Eliminates the need to continuously specify full pathnames for files and directories
  - “Relative pathnames” are locations worked out in relation to (relative to) to the **working directory**.
- “Home” directory is (usually) configured to be your working directory upon logging into the system
  - Sometimes called the “login” directory
  - /home/zastre & /home/seng265 are typical home directories

# Directory commands (2)

- traversing directories

```
% cd /usr
% ls
ucb bin lib
```

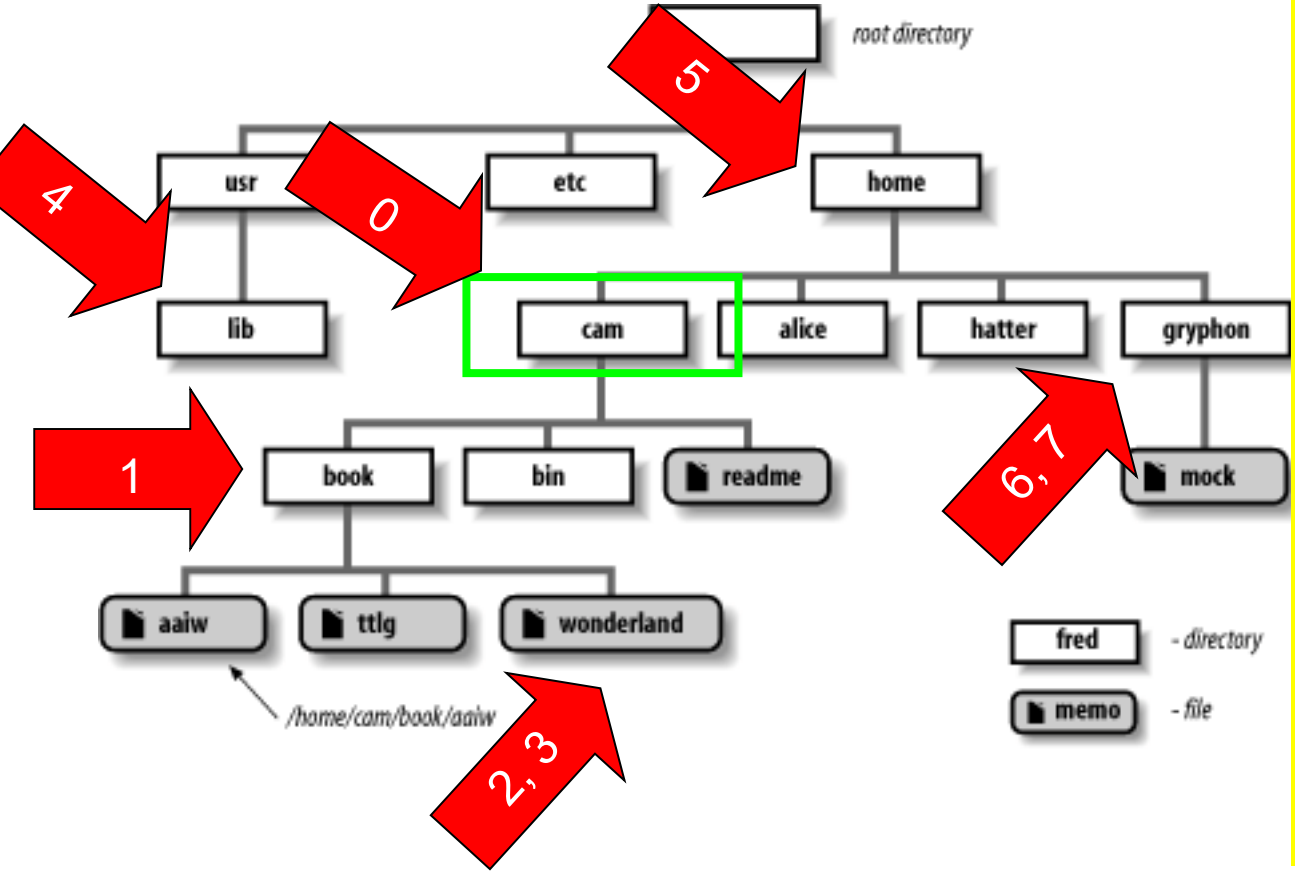
- display the **current working directory**

```
% pwd
/usr
```

- creating a **symbolic reference** to a file (i.e., like an alias)

```
% cd ~zastre
% # ln –s <target> <name of alias>
% ln –s a1 sample_solution
% ls
a1  sample_solution
```

# working directories




```
# “cam” is the logged-in user
#
# Each of the following commands
# starts in Cam’s current directory
# /home/cam (i.e., every item
# below assumes we reference
# from at red-arrow 0).
% cd book                  #1
% vi book/wonderland       #2
% vi ~/book/wonderland     #3
% cd /usr/lib              #4
% cd ..                    #5
% cd ../gryphon            #6
% cd ~gryphon              #7
% cd alice                 # ??
```

# File attributes

- every plain file and directory has a set of **attributes,** including:
  - user name (owner of file)
  - group name (for sharing)
  - file size (bytes)
  - creation time, modification time
  - file type (file, directory, device, link)
  - permissions

```
% ls –l unix.tex test
-rwxr-xr-x  1  joe  users    200  Dec 29  14:39  test
-rw-r--r--  1  dmg  users   21009  Dec 29  14:39  unix.tex
```

# Who has permission?

- permissions can be set for
  - user ("u") [-rwx------]: the file owner
  - group ("g") [----rwx---]: group for sharing
  - other ("o") [-------rwx]: any other
  - all ("a"): user + group + other
- **user**: the owner of the file or directory; owner has full control over permissions
- **group**: a group of users can be given shared access to a file
- **other**: any user who is not the owner and does not belong to the group sharing a file

# What kind of permissions?

- files:
  - **read (r)** : allows file to be read
  - **write (w)**: allows file to be modified (edit,delete)
  - **execute (x)**: tells UNIX the file is executable
  - **dash (-) :** owner/group/other have no permissions
- directories:
  - **read (r)**: allows directory contents to be read (listed)
  - **write (w)**: allows directory contents to be modified (create, delete)
  - **execute (x)**: allows users to navigate into that directory (e.g, with the **cd** command)
  - **dash (-)** : owner/group/other have no permissions

# chmod: set file permissions

- there are several ways to use "chmod"
  - use letter symbols to represent "who" and "what"

    % chmod o+rx ~/.www/ppt  # other can read and cd "ppt"
    % chmod u+x run.pl  # script "run.pl" executable
    % chmod go-rwx ~/private  # removing access group & other
    % chmod u=rwx,g=rx,o=x foobar.txt  # all permissions
  - can also use "octal" (base 8) notation, representing each three-bit field with an octal digit; $r \in \{0,4\}$, $w \in \{0,2\}$, $x \in \{0,1\}$

    % chmod 751 foobar.txt  # specify all permissions
  - the following are different ways of setting "read-only" permission for a file

    % chmod =r file
    % chmod 444 file
    % chmod a-wx,a+r file

# Various & Sundry

- UNIX file names are case-sensitive
  - assumption here: underlying file system is UNIX
  - e.g., myFile and myfile are two different names, and the logout command cannot be typed as Logout
- commands are available to change the **owner** and/or **group** of a file; e.g. chown, chgrp
- **pager** is a command (less, more) used to display a text file one page at a time

  % less unix.txt
- quickly create a file (or update the timestamp of an existing file)

  % touch unix.txt

  % ls –l unix.txt

  -rw-r--r-- 1 zastre users 0 Aug 29 14:39 unix.tex

# the shell (again)

- The shell is **the** intermediary between you and the UNIX OS kernel
- It interprets your typed commands in order to do what you want
  - the shell reads the next input line
  - it interprets the line (expands arguments, substitutes aliases, etc.)
  - performs action
- There are two families of shells:
  - “sh” based shells, and “csh” based shells
  - they vary slightly in their syntax and functionality
  - we’ll use “bash”, the Bourne Again SHell (derivative of “sh”, known as the “Bourne shell”)
  - tip: you can find out what shell you are using by typing:
    echo $0

# basic command syntax

% cmd [options] [arguments]

- cmd represents here some builtin-shell or UNIX command
- [options] = zero or many options
- [arguments] = zero or many arguments

| ***option*** | ***example*** |
|---|---|
| opt | a |
| -opt | -v |
| --optname | --verbose |
| -opt arg | -s 5 |
| --optname arg | --size 5 |

# basic command syntax (2)

- opt is a character in {a..zA..Z0..9}
- optname is an option name； e.g.， --size, --keep
- argument, arg is one of the following：
  - file name
  - directory name
  - device name, e.g., /dev/hdb2
  - number, e.g., 10, 010, 0x1af, …
  - string, e.g., "*.c", "Initial release"， …
  - …

# command types

- commands can be:
  - built into the shell (e.g., cd, alias, bg, set,…)
  - aliases created by the user or on behalf of the user (e.g., rm='rm –i', cp='cp –i', vi='vim')
  - an executable file
    - binary (compiled from source code)
    - script (system-parsed text file)
- Use the type command to determine if a command is builtin, an alias, or an executable.

```
% type rm
rm is aliased to ‘rm –I’
```

# some simple commands

% cat [file1 file2 ...]

- (catenate) copy the files to stdout, in order listed

% less [filename]

- browse through a file, one screenful at a time

% date

- displays current date and time

% wc [filename]

- (word count) counts the number of lines, words and characters in the input

% clear

# getting help on commands

- You can ask for help in several ways.
- Display a long description of a command (from section ***n*** of manual)

  % man [n] chmod
- Display a one line description of a command

  % whatis gcc

  gcc  gcc (1) - GNU project C and C++ compiler
- Use "info"

  % info gcc

  % info ls
- Many commands provide their own help

  % somecmd –h

  % somecmd --help