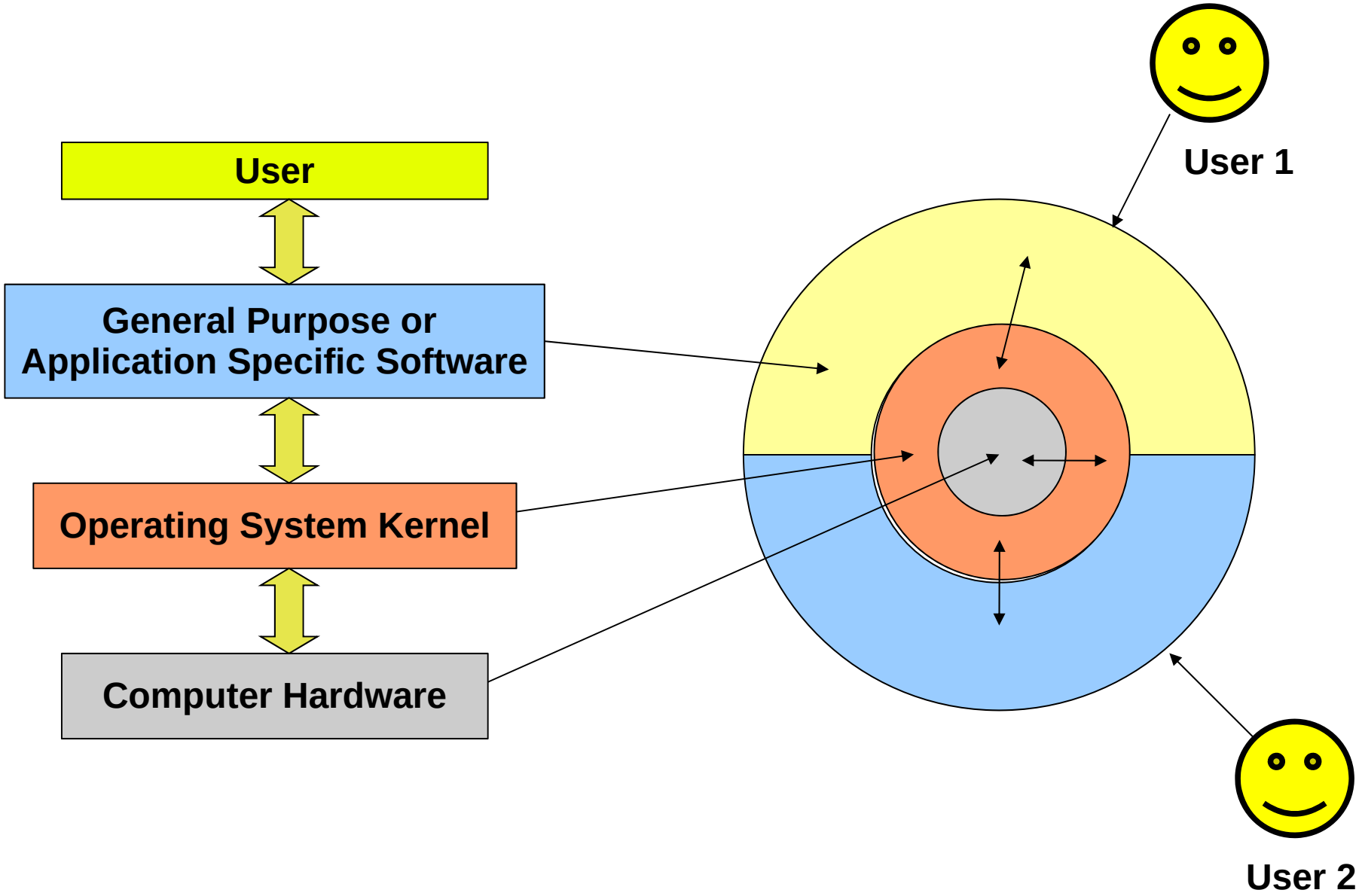
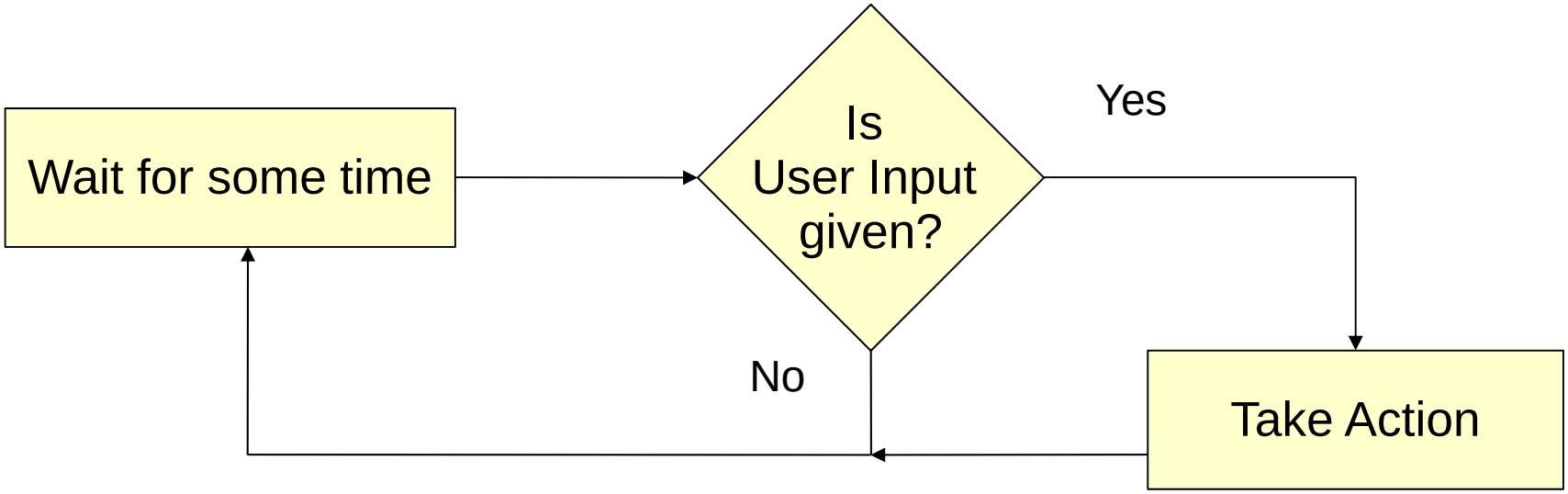
The Linux Command Line

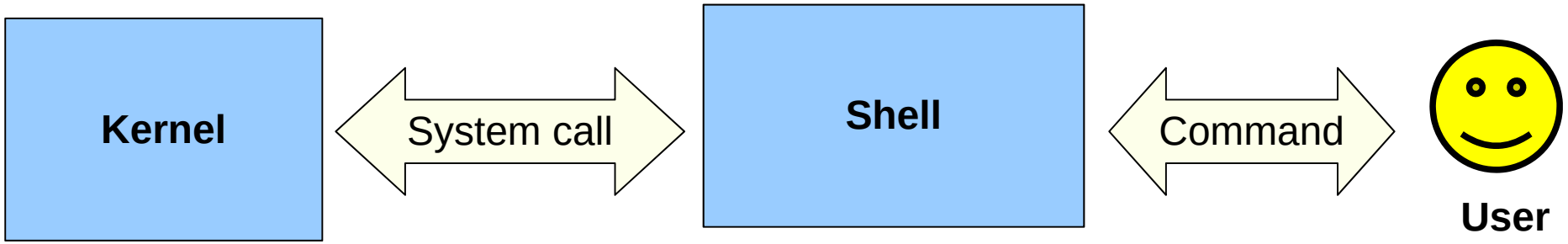
The Operating System



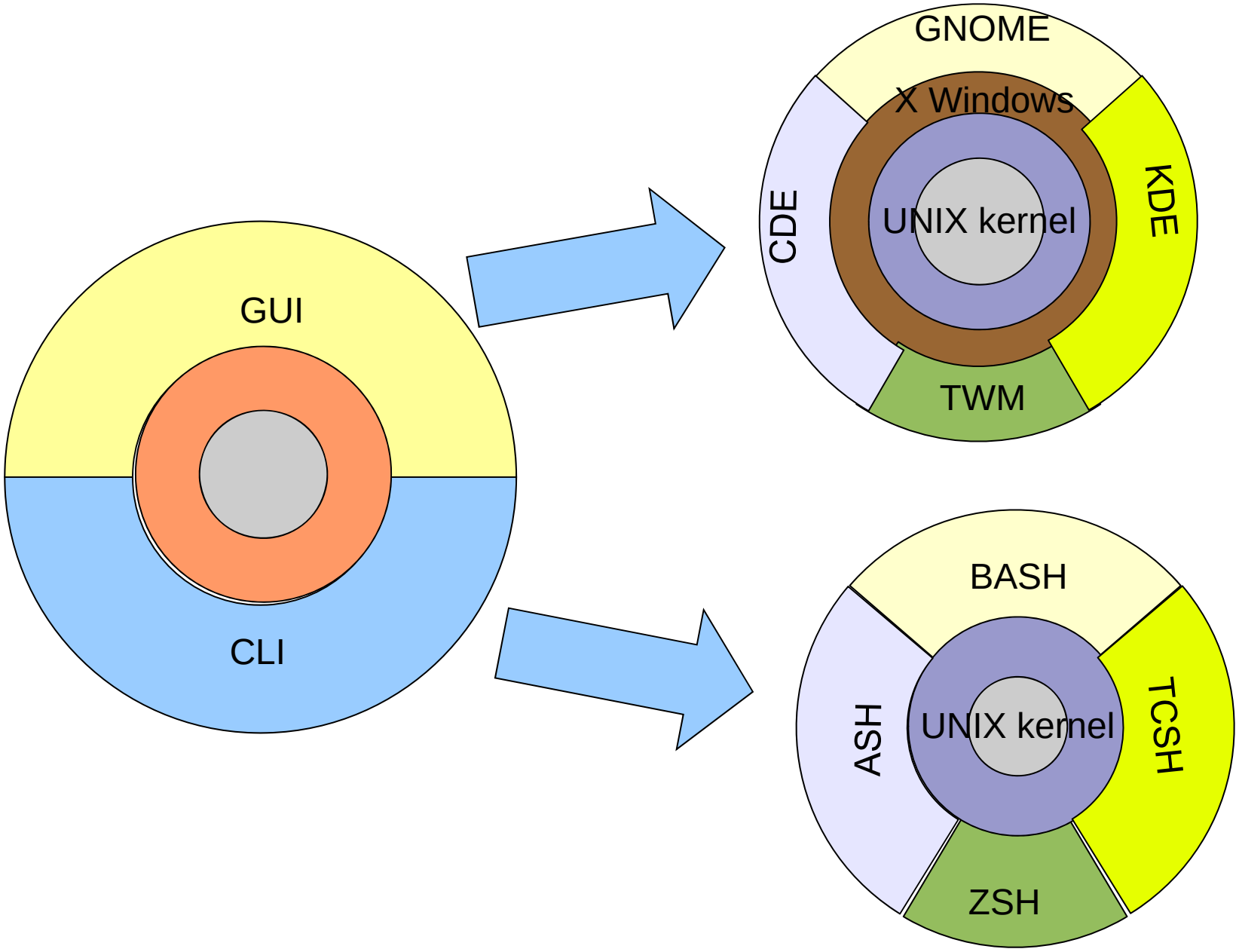
The Shell

The shell is a command interpreter.





UNIX User Interfaces/Shells



I've got the Power!!

The following example underscores the power of the UNIX Command Line:

- I want to change all variable names in C++ source and header files from m_<NAME> to <NAME>_ and save the original files as <filaname>.bak:

```
$ find . -name '*.cpp' -o '*.h' -exec cp {} {}.bak \; -exec
sed -n "s/m_\([A-Za-z]*\)/\1_/gw {}" {}.bak \;

$
```

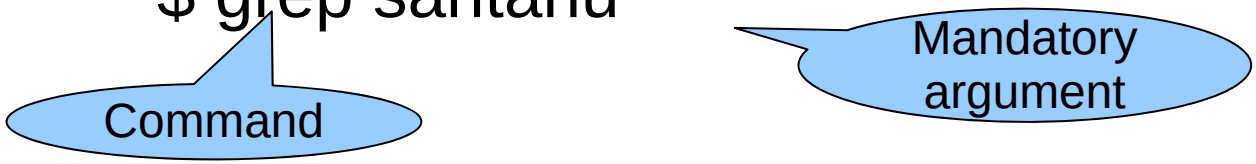
Structure of an UNIX command

- For each executable command atleast the following are necessary to be given to the command execution shell:
  - The command name itself
  - An list of “option”-s. These can modify the behavior of the program.
  - Other arguments necessary for the execution of the program.
    - For example a command to show the contents of a text file will need to have the file name as it's argument from the command line.
    - Such arguments may be mandatory or optional.

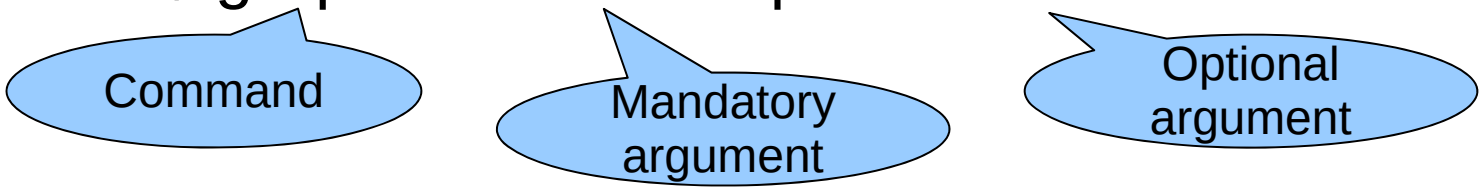
Example of an UNIX command

- Example:

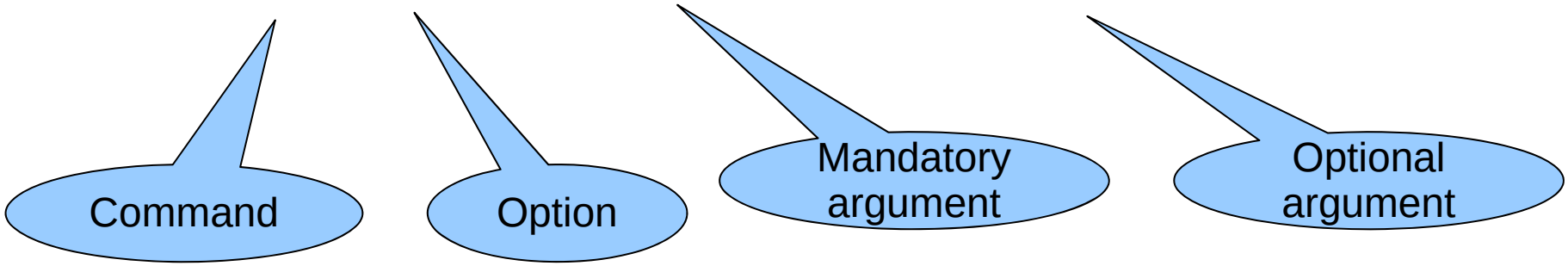
$ grep santanu



$ grep santanu /etc/passwd



$ grep -v santanu /etc/passwd



man: The manual viewer

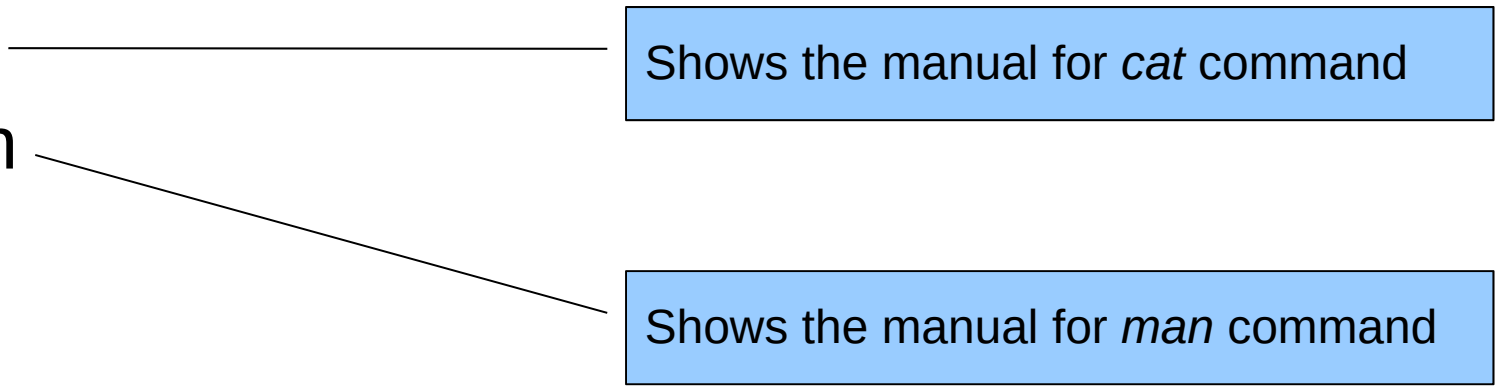
- The *man* command can be used to view the user manual on an UNIX system.

Syntax: man <command>

Example Usage:

$ man cat

$ man man



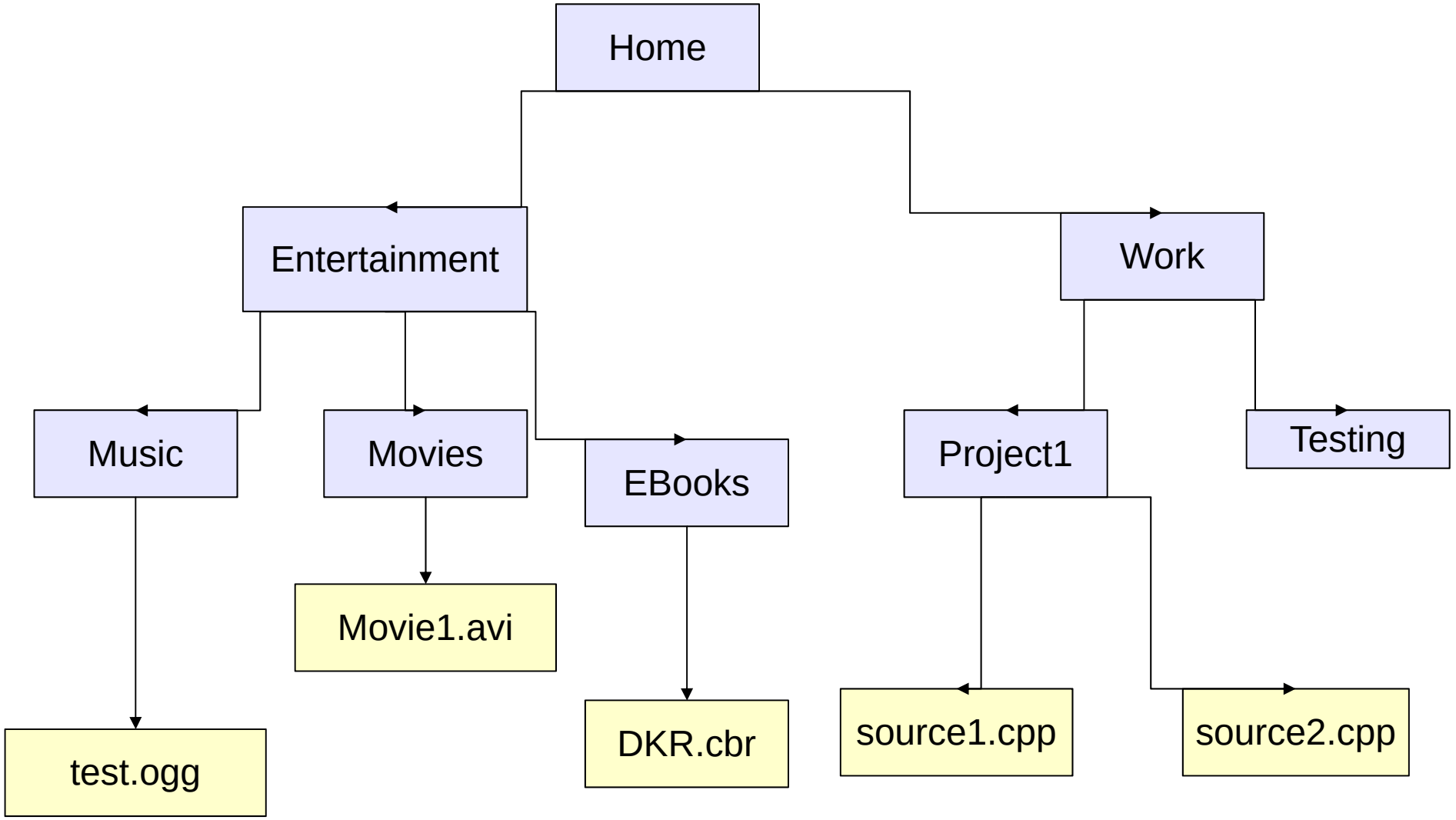
Files

- Major types of files:
  - Text: Represented as a stream of bytes. Represented in ASCII or Unicode.
    - Example: C source files, System configuration files etc.
  - Binary: Representation is dependent on application.
    - Example: MP3 files, AVI files etc.
  - Special Files:
    - Directories: Logical collection of files
    - Device Files: Not actually files, but file like interfaces to devices in and/or attached to the system. Most of these can be programatically manipulated using system calls similar to manipulation of text files.
      - Example: /dev/rfcomm0, /dev/tty ...
    - Others

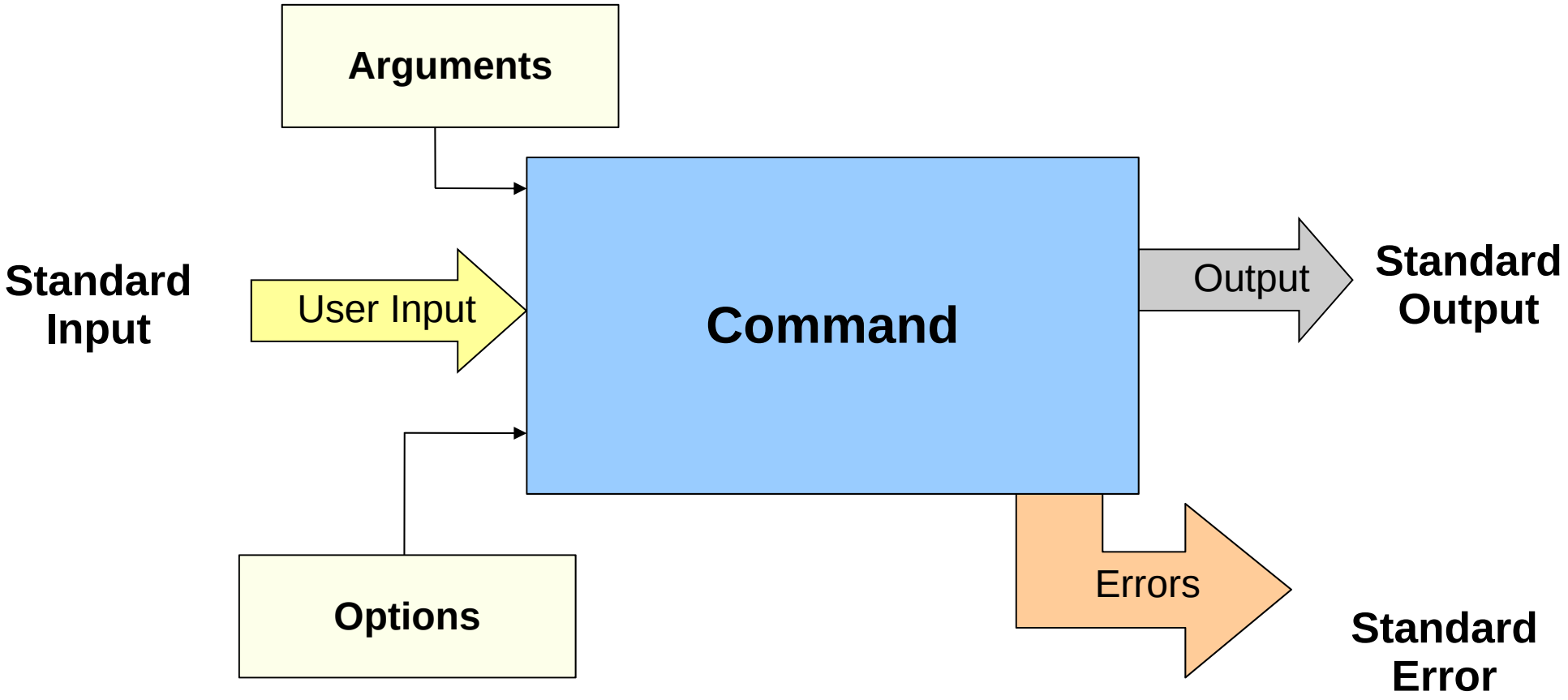
Directory

- A directory is a logical collection of files and other directories.
- Directories make life easier by allowing for easy maintainability and storage of files on a system.
- User is logged into the use's “Home” directory
- All user created files will reside in the directory “tree” inside user's home directory.

An user's typical home directory



The standard input, output and error streams



Facilities from the Shell

- Meta-characters:
  - * : Match all
  - ?: Match one
- Input/Output redirection
  - < : Get input from file
  - <<: Here Document
  - > : Send output to someplace
  - >> : Apped output
  - | : Send output to stdin of some command

A basic set of text file manipulation commands

- file
- vim
- cat
- pr
- less
- diff
- merge

file: Know file type

File command can be used to inspect the type of of a file.

Example usage:

$ file test.c

test.c: ASCII C program text

$file test.mp3

test.mp3: Audio file with ID3 version 24.0 tag, MP3 encoding

vim: The editor

- Can be used to create, edit and view muliple text files at the same time.
- Provides standard file editing features such as search and replace, undo/redo, copy/paste etc.
- Provides advanced editing capabilities for editing source code. Examples include: syntax highlighting, automatic indentation, auto-completion etc.
- Invocation:

  $ vim

  $ vim filename

  $ vim filename1 filename2

vim: Modes of operation

- *Command mode:* Receives commands to do various tasks such as copy/paste text, save files and so on.
- *Ex mode:* Takes various commands native to the “ex” line editor. Extremely useful for various operations such as complex search/replace, Saving file to other name etc.
- *Edit/Replace mode:* In this mode we write the contents of a file or replace the older contents with newer ones.

vim: Moving from one mode to another

- ***<Esc>***: Move to command mode from Ex/Edit mode
- ***:*** : Move from command mode to Ex mode
- ***I*** or ***<INSERT>***: Move from command mode to Edit mode
- ***<INSERT><INSERT>***: Move from command mode to replace mode

vim: Basic editing command keys

- **Command mode:**
  - .: *Repeat last action*
  - *ZZ:* Save file and exit
  - *y[y/w/$]:* Copy line/word/upto end of line to buffer
  - *d[d/w/$]:* Delete/Cut line/word/end of line to buffer
  - *p: Paste last copied/deleted text from buffer after current line/position*
  - *P: Paste last copied/deleted line from buffer before current line/position*

  All copy and pase operations can be preceeded by an integer n to identify the number of times this task is to be repeated
- ***Ex mode:***
  - *w [filename]: save file [to name]*
  - *e [filename]: Open file for editing*

cat: Show files

- Can be used to see the contents of a file without having to open it in vim.
- Able to show multiple files one after another
- If no options are given, it reads the command line.

Examples:

$ cat a.txt	Shows file a.txt
$ cat a.txt b.txt	Shows file a.txt and b.txt
$ cat	Reads from stdin

pr: Pretty printer

- Formats page for printing
- If no argument is given, it reads from the command line

Example:

$ pr a.txt	Formats a.txt
$ pr +2 a.txt	Prints page 2 only
$ pr +2:3 a.txt	Prints page range 2 to 3
$ pr -2 a.txt	Prints a.txt in 2 columns

Using “|” to join comands (Contd.)

- Example:

$ cat a.txt c.txt > b.txt

$ pr b.txt

Combined:

$ cat a.txt c.txt | pr

less: View files

- For viewing files on the screen one page at a time
- Allows forward and backward movement
- Much faster than vi
- Eithes filenames are to be given, or it takes input from pipe
- Example

```
$ less a.txt

$ cat a.txt b.txt | less
```

diff: See the difference

- Shows the difference between files
- Giving atleast two file-names are necessary

Example:

$ diff a.txt b.txt

See difference between a.txt and b.txt

Directory management commands

- ls
- cd
- mkdir
- rmdir
- cp
- mv
- rm

pwd: Present working directory

- See the directory where the user is currently placed in
- After login, the output of the pwd command shows user's home directory

Example:

```
$ pwd
```

ls: List the contents

- Shows the contents of a directory
- If no arguments are given, it shows the current directory
- Important options:
  - ***-l :*** **Long format directory listing**
  - ***-R:*** **List directories recursively**

Example:

$ ls -l	**Shows current directory in long format**
$ ls Music Movies	**Shows contents of Music and Movies**

mkdir: Create a directory

- Used to create directories
- With -p option it can create entire tree of directories

Example:

$ mkdir dir1 **Create a directory called dir1**

$ mkdir dir1 dir2 **Create directories dir1 and dir2**

$ mkdir -p dir1/dir2 **Create dir1 and create dir2 inside dir1**

rmdir: Remove a directory

- Used to remove *empty* directories
- With -p option it can remove entire tree of directories

Example:

$ rmdir dir1 **Remove a directory called dir1**

$ rmdir dir1 dir2 **Remove directories dir1 and dir2**

$ rmdir -p dir1/dir2 **Remove dir2 then remove the parent dir1**

Some special directories

- The following special directories can be used:
- . : The “dot” means the current directory
- .. : The “dot dot” means the parent directory
- ~: The “tilda” means the user's home directory

cd: Move around

- Used to walk around in the directory tree

Example:

$ cd dir1	**Enter a directory called dir1**
$ cd dir1/dir2	**Enter dir2 which is inside dir1**
$ cd ..	**Enter the parent directory**
$ cd ../dir2	**Enter dir2 which is above the current directory**
$ cd ~	**Enter the user's home directory**
$ cd ~/dir1	**Enter dir1 which is in the home directory**

cp: Copying files and directories

- Copies file/directories from one directory to another
- The -r option is required when copying directories.

Example:

$ cp a.txt dir2 **Copies a.txt to dir2**

$ cp dir1/a.txt dir2 **Copies a.txt to dir2**

$ cp ~/dir1/a.txt dir2 **Copies a.txt from dir1 in home to dir2**

$ cp ~/dir1/a.txt .

Copies a.txt from dir1 in home to pwd

$ cp -r ~/dir1 . **Copies dir1 from home to current directory**

mv: Move files and directories

- Moves files and directories from one place to another

Examples:

$ mv ~/dir1 .

Moves directory dir1 from home to the current directory

rm: Remove files and directories

- Can be used to remove files
- -r option is required to delete directories
- -f option forces deletion

Example:

$ rm a.txt b.txt **Removes a.txt and b.txt**

$ rm -r dir1 a.txt **Removes dir1 and a.txt**

Filter commands

Simple filters

- cut
- tr
- grep
- head
- tail
- sort
- uniq
- wc

Reconfigurable filters

- sed
- awk

head: See the top of a file

- Can be used to view the first few lines
- By default 10 lines are displayed
- -n option displays the first n lines
- If no filename is given then it reads from stdin

Example:

$ head foo.txt **Display first 10 lines of foo.txt**

$ head -5 foo.txt **Display the first 5 lines of foo.txt**

$ cat foo.txt | head -2

tail: See the end of a file

- Can be used to print the last few lines of a file
- By default last 10 lines are displayed
- -n option prints the last n lines
- If no filname is given then reads from stdin

Example:

$ tail foo.txt **Display last 10 lines of foo.txt**

$ tail -5 foo.txt **Display the last 5 lines**

$ cat foo.txt | tail -2

wc: Word count

- Count the number of words, lines and/or characters
- -l option shows number of lines
- -w shows number of words

Example:

$ wc foo.txt **Display last 10 lines of foo.txt**

$ wc -l foo.txt **Display the last 5 lines**

$ cat foo.txt | wc -l

tr: Change a character

- Translates one character to another
- -s: squeeze multiple characters into one
- -d: delete all occurances of a particular character

Example:

```
$ cat abc.txt | tr 'a' 'A'

$ cat abc.txt | tr '[a-z]' '[A-Z]'

$ cat abc.txt | tr -s 'a'

$ cat abc.txt | tr -d 'a'
```

cut: See columns

- Select columns from output
- -d: Specify delimiter between fields
- -f: Specify filed number(s) to view

```
$ cat abc.txt| cut -d ':' -f 1
$ cat abc.txt| cut -d ':' -f 1-2
$ cat abc.txt| cut -d ':' -f 1,4-
```

sort: Sort the output

- Sorts thee output aplabetically or numerically
- -n: Numeric sort
- -r: Reverse sort
- -u: Sort and find only uniq

```
$ cat abc.txt|sort
$ cat abc.txt|sort -n
$ cat abc.txt|sort -rnu
```

Regular Expressions

- Pattern that describes a set of strings
- Important characters:
  - []: Provide a set of alternative characters
  - ^/$: Position markers
  - \ : Escape
  - ?/*/+/{n}: Repitition specifier
  - \n : Back-reference
  - | : or 2 regular expressions

grep: Find a regular expression

- Prints lines where expression is found
- -v: Print non-matching lines
- -r: Search recursively
- -c: Count number of occurances
- -n: Print line number

```
$ cat abc.txt|grep '^error'
```

```
$ cat abc.txt|grep -v '^error'
```

```
$ grep -r error src/
```

sed: Stream editor

- Filters and transforms text from streams
- Idea is to find a pattern and do some action on that
- Example:

```
$ cat abc.txt | sed -n '/hello/p'
$ cat abc.txt | sed '/hello/d'
$ cat abc.txt | sed '/[hH]ello/s//Hi/g'
$ cat abc.txt | sed '1,10d'
$ cat abc.txt | sed '1,/hello/p'
```

awk: Find something, do anything

- Pattern scanning and proessing language
- Example:
  - Basic syntax:

    $ cat abc.txt | awk '/hello/{print}'
  - Special options:

    $ cat abc.txt | awk 'BEGIN{ print “start”} \

    > /[hH]ello/{print $NF} \

    > END{ print “end” }'

awk: Contd.

- A simple word counting tool
  - File:

```
BEGIN { count=0 }
{ count += NF; }
END { print "Word Count:" count }
```

  - Running:

```
$ cat abc.txt | awk -f sample.awk
Word Count: 8
```

chmod: Change permissions

- Change permissions on a file
- Examples:

```
$ chmod go+r abc.txt
$ chmod a+rx count.sh
$ chmod 755 count.sh
$ chmod go-rwx personal_stuff
```

find: Find files and folders

- Search for files and directories in the specified path and apply some action
- -name '<regexp>' - match filenames matching regexp
- -type f/d - find all entries of type file(f) or directory(f)
- -exec *command* -execute command on found entry
  - The entry is enoted by { } in the cmmand
- -print – print the found entry
- -a/-o – And/or

find: Contd.

- Example:

```
$ find . -name '*.c' -print
$ find . -type f -print
$ find . -type f -a -name 'abc*' -print
$ find . -name 'abc' -exec wc -l {} \; -print
$ find . -name '*.mp3' -exec cp {} . \; -print
```

A basic shell script

- The file:

```
#!/bin/tcsh
# This script prints hello world
echo “Hello World”
exit 0
```

- Change permission:

```
$ chmod +x hello.sh
```

- Run:

```
$ ./hello.sh
Hello World
$
```

echo: Show a string

- Prints the provided strings
- -n option skips the newline addition
- Example:

  $ echo “Hello World”

  Hello World

  $ echo Hello World

  Hello World

  $ echo -n Hello World

  Hello World$

Variables

- A container for text
- Variable name can be alphanumeric
- Value can be accessed as:
  - **Read:** $variablename or ${variablename}

    $ echo ${x}
  - **Write:** variablename

    $ set x=10

    $ set x=$<

Say hello in style

- The program:

```
#!/bin/tcsh
echo -n "Enter name:"
set name=$<
echo "Hello ${name}"
exit 0
```

Special variables: PATH

- PATH variable stores the installation paths of all executables in the system.
- For adding your own path to the default path use the setenv comand.

Example:

```
$ echo $PATH

/opt/j2sdk1.4.2_15/bin:/usr/local/bin:/usr/bin:
    /bin:/usr/bin/X11:/usr/games:/usr/local/bin:/usr/bin:/bin:/usr/gam
    es

$ setenv PATH ${PATH}:/home/santanu/bin
```

Command line parameters

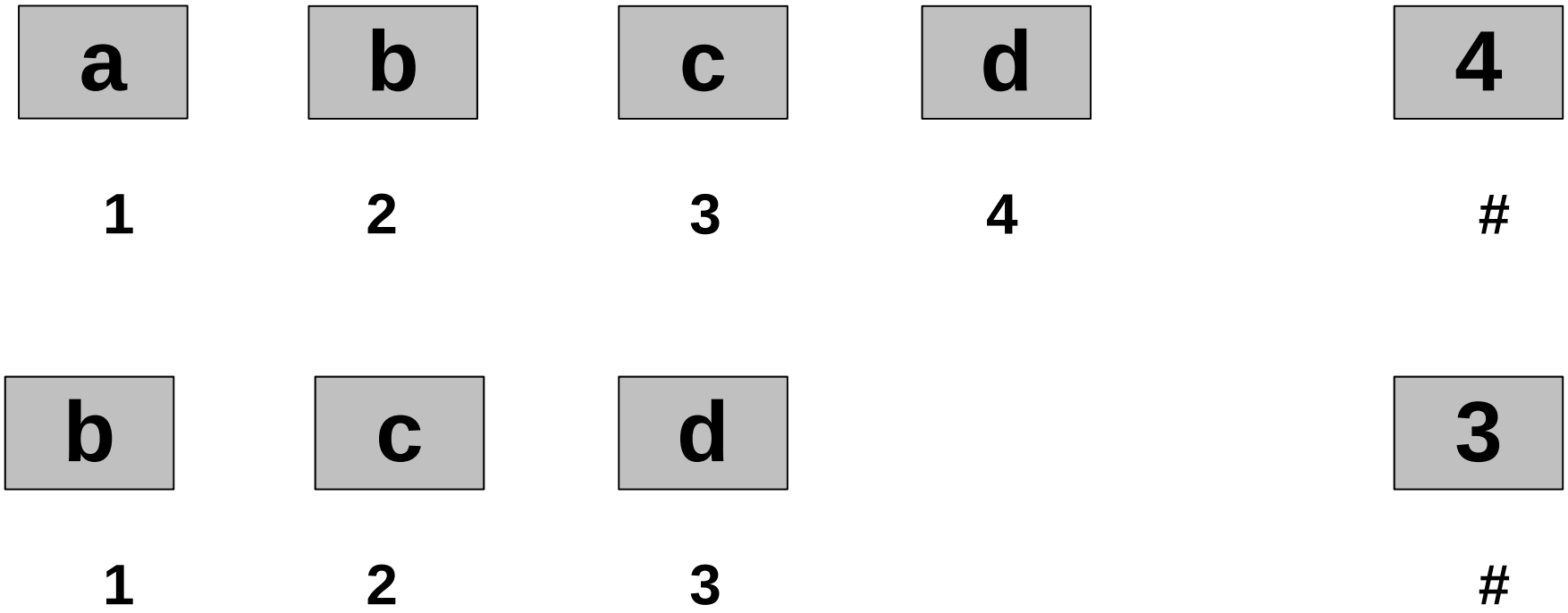
- Arguments to command line are available in special variables
- Variable names start with 0
- ${0} given the script name itself
- Other parameters are available in variables starting with 1 (access value as ${1}) and so on
- Total number of command line parameters can be found in $#

Command line parameters (contd)

```
#!/bin/tcsh
echo “Script name: ${0}”
echo “$# arguments have been supplied”
exit 0
$ ./first.sh
Script name: ./first.sh
0 arguments have been supplied
$ ./first.sh aa bb
Script name: ./first.sh
2 arguments have been supplied
$
```

shift: Move command line arguments

- Shift command will move argument 1 out of buffer and move $2 to 1, $3 to 2 and so on
- $# is updated accordingly



shift (contd)

```
$ cat second.sh
#!/bin/tcsh
echo "First argument is ${1}"
shift
echo "First argument is now ${1}"
exit 0
$ ./second.sh a b
First argument is a
First argument is now b
$
```

/dev/null and $status

- Useful when we want to know only whether a command succeeded or not
- ***/dev/null*** file eats up any input given to it
- ***$status*** (or ***$?***) gives the return value of the last command

Example:

```
$ grep echo first.sh >/dev/null
$ echo $status
0
$ grep hocuspocus first.sh >/dev/null
$ echo $status
1
$
```

test and *[ ]*: test for something

- Test for certain conditions, returns success if expression evaluates to true
- Commonly required conditions:
  - -e FILE
  - -f FILE
  - -d FILE
  - Op1 -eq Op2
  - Op1 -ne Op2
  - Op1 -le Op2
  - Op1 -ge Op2
  - Op1 -lt Op2
  - Op1 -gt Op2
  - str1 == str2
  - str1 != str2
  - -n str
  - -z str

test: Contd

```
$ set x=10
$ [ $x -eq 10 ]
$ echo $status
0
$ [ $x -eq 11 ]
$ echo $status
1
$ test $x -eq 10
$ echo $status
0
$ test $x -eq 11
$ echo $status
1
$
```

expr: Do the math

- Evaluates common mathematical expressions
- Common operators:
  - +, -, *, /

Example:

```
$ expr 2 + 3
5
$ expr 3 – 2
1
$ x=2
$ expr $x + 1
3
```

`(back-quote): Command substitution

- Substitute the enclosed command with the screen output as obtained by running the command

Example:

```
$ set dateval=`date`
$ echo ${dateval}
Sat Mar 29 21:33:01 IST 2008
$ set x=2
$ set y=3
$ set z=`expr ${x} + ${y}`
$ echo $z
5
$
```

If: Simple branching

- **Syntax:**

```
if condition then
    # take necessary actions
else
    # take necessary actions
endif
```

- **Numbers are compared with the conventional relational operators (== != < > <= >=)**
- **Strings are compared with =~ and !~**
- **Otherwise use *test***

If: Continued

```
$ cat third.sh
#!/bin/tcsh
if ( $USER =~ debray ) then
  echo Nice guy!
else
  echo Who are you?
endif
$ ./third.sh
Nice guy!
$
```

case: Multiway branch

- **Syntax:**

```
switch (word) in
    case pattern1:
        command list
        breaksw
    case pattern2:
        command list
        breaksw
    ...
endsw
```

case: Contd

```
$ cat fourth.sh
#!/bin/tcsh
switch ($#)
case 0:
    echo "Error: string and filename(s) not given"
    echo "Usage: $0 [<string>] [filename1] ..."
    exit 1
    breaksw
case 1:
    echo "Error: filename(s) not given"
    echo "Usage: $0 [<string>] [filename1] ..."
    exit 1
    breaksw
default:
    string=${1}
    filename=${2}
endsw
echo "String: ${string}"
echo "Filename: ${filename}"
$
```

while: Loop construct

- **Syntax:**

```
while (expr)
    #do necessary actions till expr is non-zero
end
```

```
$ cat fifth.sh
#!/bin/tcsh
while ($# > 0)
  echo -n $1
  shift
end
$ ./fifth.sh a b c
abc$
```

foreach: Loop construct

- **Syntax:**

```
foreach item (item1 item2 item3 ...)
    #do necessary actions on $item
end
```

```
$ cat sixth.sh
#! /bin/tcsh
# loop through a set of numbers
foreach i (1 2 3 4 5)
  echo -n "$i"
end
$ ./sixth.sh
12345
```

foreach: Continued

```
$ cat seventh.sh
#! /bin/tcsh
# loop through a set of files
foreach f (*.html)
  echo $f
end
$ ./seventh.sh
bar.html
foo.html
...
...
$
```

A summary of concepts

- Multiuser Operating Systems
- Shell and Kernel
- Files
- Directories
- Vi editor
- Utility/File Management Commands
- Filters
- Shell Scripting

Process

- Running instance of a program in memory
- Each process has it's own unique ID called the *pid* of the process
- *ps* – Shows process listing
  - $ ps
- *kill* – kill a process
  - $ kill 1234
- <*ctrl*>-*Z* – pause and back ground a process
- *bg* – resume the process in background
- *fg* – bring the “bg”-d process to foreground

A summary of concepts

- Multiuser Operating Systems
- Shell and Kernel
- Files
- Directories
- I/O Channels of a program
- Vi editor
- Utility/File Management Commands
- Filters
- Shell Scripting
- Process

Questions?

Courtesy: Santanu Sinha and Debarshi