CS 214 / 2021-04-19
===================

Command-line tricks

    "file globs"

    * -> matches any sequence of zero or more characters
    If we use * in a command, the shell will look for files matching this pattern

    *.txt -> replaced with the names of all files that end in .txt
    foo*bar -> replaced with the names of all files that begin with foo and end with bar

    ? -> matches one character (wildcard)

        section??.txt -> matches section01.txt and section99.txt,
            but not section1.txt or section100.txt

    By default, globs work in one directory (the working directory)
    With /, we search in different directories

    *.c   <- matches all C source files in the current directory
    src/*.c <- all C source files in the subdirectory src
    ../*.c   <- all C source files in the parent directory

    /.c   <- all C source files in any subdirectory
    ../*/*.c   <- all C source files in any subdirectory of the parent directory

    *.c
    /.c
    //*.c

    ../src/module*/*.c
    /usr/include/lib*/*.h

    The shell replaces globs with lists of file names

    If we type this
        mv *.h include

    The shell replaces it with
        mv foo.h bar.h baz.h include

    mv *.c *.bak <- sadly not possible
        <- mv never sees the globs, so it can't tell what we want to do

Shell scripts

    Just bunch of shell commands in a (text) file
    Mark as a executable
    begin with #! followed by a path to the interpreter

    Executing a shell script is just as if you had entered the commands yourself

---
#!/bin/bash

cp *.h backups
cp *.c backups

    ./mybackup.sh

Useful Unix commands

cat
    - "concatenate"
    - reads one or more files and prints to stdout

    cat section*.txt
        - combine all files whose names match the pattern and print
    cat section*.txt > output
        - combine all files and write to a file named "output"

more
    - like cat, but pauses each time the screen fills up
    - related program: less
        - does the same thing, but is fancier

    - can give multiple arguments, or no arguments to read from stdin

        -> we can pipe the output of a program to more, and let more present it
            one screenful at a time

    ps -ef   <- dumps information about all running processes
    ps -ef | more   <- dumps the same information, one screen at a time

head [-number of lines] [files...]
    - prints the first few lines of a file (or stdin)
    - can optionally specify the number of lines (e.g. -20 to get 20 lines)

tail [-number of lines] [files...]
    - prints the last few lines

    Recall: ls -ltr lists all files in reverse chronological order
        ls -ltr | tail   <- only prints the last few
        ls -ltr | tail -1   <- only print the most recently modified file

file [files...]
    - guess what kind of data a file contains
    - uses a variety of methods, including looking for format markers
        and doing statistical analysis
    - purely for your assistance; no programs depend on this output
    - does not look at file names (extensions mean nothing)

wc [options] [files...]
    - "word count"
    - counts the number of characters, words, and lines in a file or files
    - can request only certain counts (e.g., -l for just the number of lines)
    - reads from stdin if no arguments

sort [files...]
    - sorts lines alphabetically
        - concatenates all input files, and/or reads from stdin

uniq
    - reads its input, but skips any line that is the same as the previous line
    - read from multiple files and/or stdin
    - use -c to print the number of times a line is repeated

grep [options] [pattern] [files...]

    - multi-file text search using regular expressions
        - look for lines matching the pattern and print them
    - can read from multiple files and/or stdin
        -> can use grep to search the output of a program
            ps -ef | grep lab
            ps -ef | grep -v root

    regular expression syntax
        * repeat zero or more times
        + repeat one or more times
        . wildcard
        [abcd] match any listed character
        [a-z]   match any listed character in the given range
        ^   match start of line
        $ match end of line

        [0-9]+ match any sequence of decimal digits

    -> not the same syntax as file globs!

    -v negates the grep (print the lines that do not match)
    -c counts the number of lines that matched (per file)
    -n prints file name and line number before the match

    -> many options! so many!

cmp [file1] [file2]
    says whether two files are the same
        EXIT_SUCCESS and no output if the same
        EXIT_FAILURE and some output if different

diff [file1] [file2]
    compare two files and print all lines that are different

    -> frequently used to compare versions of a source file

    -> can use this to compare the output of a program to its expected output

        ./prog | diff expected -

    - many options to control what is considered a difference and how it is reported

worth looking into: sed, awk

ps
    - lists running processes
    - use -e to get all processes
    - many options to get more details, such as -f

top
    - lists all running processes, sorted by CPU use
    - updates the screen: live listing

Networks & inter-process communication

broadly: how do we send a message from one computer to another?
    - more precisely, from one process to another

there are many ways for processes on the same computer to communicate
    -> file system
    -> pipes
    -> use the OS to pass messages
    -> shared memory

    usually require both processes to be on the same computer
        frequently, one process must be the parent of another

We want communication between processes
    - running on different devices (no shared memory, disks, etc.)
    - started independently of each other

Questions
    - how is the communication organized
        - send individual messages?
        - stream of bytes?
    - how are messages transferred from one process to the other?
    - how do the processes identify each other?

Any networking system must be able to answer these questions
    - there have been many networking systems that have had different answers

Typical designs are layered
    different subsystems have different areas of responsibility
    standard interfaces used to access these systems
        -> sockets are one such interface

socket interface is very general
    -> we can use the same interface for many different kinds of network
    -> a typical program will only use a small number of networks (e.g., one)