Slide 1

Administrivia

- Reminder: Homework 2 due Wednesday.

Slide 2

The Big Picture, Again

- Material in this course can come across as a bunch of parlor tricks — fun in their way, but “so what?”
- The “big picture” view — introduce you to a range of tools that can help you “work smart, not hard”. (“Laziness in programmers is a virtue”?)

  The idea — if it’s tedious and repetitive and can be done by the computer rather than by the human, make the computer do it! even if that requires the human to think a bit more.

  Once you start thinking along these lines, you may work differently with other tools too (using keyboard shortcuts rather than menus, cutting and pasting rather than retyping, etc.).

Slide 3

Pipes — Review

- “Pipes” provide one-way communication between programs — output of program A becomes input of program B.
- Key component of “the UNIX philosophy” — emphasis on providing a toolkit of small programs, mechanisms for combining them.
- “Filters” are programs designed to work this way, and there are lots of them (some in next slides and next time). `less` and `more` also useful.

Slide 4

Filters

- `head`, `tail`.
- `sort`, `uniq`.
- `grep` — search for text (or regular expression — more later).
- `wc` — count characters, words, lines.
- `tr` — “translate”. Good for converting, e.g., upper-case to lower-case.
- `tee` — duplicates input. Good for capturing output to a file while also displaying it onscreen.

Filters, Continued

- `sed` — “stream editor”. Example — convert DOS/Windows-style text file (each line ends with `\r\n`) to UNIX-style (each line ends with `\n`).
- `awk` — “pattern scanning and processing language” — many interesting possibilities; simplest is just to break up input into whitespace-delimited fields.

Slide 5

Examples

- Find all processes that belong to your username:

```
ps aux | grep $USER
```

- Generate a list of machines that are “up”:

```
ruptime | grep up | awk '{print $1}'
```

- Show how much space each subdirectory of your home directory is using, sorted by size.

```
du -sk $HOME/* | sort -n
```

(Unfortunately this omits directories starting with a dot.)

Slide 6

Slide 7

More Filters

- `sed` — "stream editor" — non-interactive program, by default does *not* edit in place, but works as a filter, transforming input to produce output. Especially useful with regular expressions (later), and in manipulating variables within a command (later).
- Some simple uses (with commands inline):
  - Search and replace:

    ```
    sed 's/old/new/g' infile > outfile
    ```
  - Delete lines containing some string:

    ```
    sed '/this/d' infile > outfile
    ```

    (How else could you do this?)

  For more complicated edits, can put command(s) in a file rather than inline.

Slide 8

More Filters, Continued

- `awk` — implementation of programming language AWK — "pattern scanning and processing language".
- Lines of AWK program specify pattern and action. (Can also include function definitions.)
- Basic processing — split each line of input ("record") into "fields", compare to patterns in program, execute actions for any patterns that match.
- Some simple uses (with commands inline):
  - Print selected fields from input (as in examples from last time).
  - Print selected lines of input:

    ```
    awk '/this/' infile
    ```

    (How else could you do this?)

  For more complicated edits, can put command(s) in a file rather than inline.

Slide 9

More Useful Commands

- `find`. Very powerful/flexible, though if you don't use it often you probably will have to read the man page to remember syntax.
- Simple examples:
  - Find all files in the current directory modified in the last week.
    ```
    find . -mtime -7
    ```
  - Find all files in your home directory whose name contains `hello`.
    ```
    find $HOME -name "*hello*"
    ```
  - Find all files in the current directory and subdirectories that end in `.bak` and remove them.
    ```
    find . -name "*.bak" -exec rm {} \;
    ```
    (The `-i` flag doesn't work in this context, but if you want to be prompted, replace `-exec` with `-ok`.)

Slide 10

More Useful Commands, Continued

- `diff` — compare files or directories. (A good use — "regression testing" of programs.)
- `pushd`, `popd` (actually shell built-ins) — manipulate shell's stack of directories.
- `cat` (concatenate — one or more inputs to output). Sometimes used when it doesn't need to be, as a substitute for redirecting input ("Useless Use Of Cat (UUOC)").

More Useful Commands, Continued

- `xargs` — "build and execute command lines from standard input".
  - Find all processes for program `java` and kill them:

```
ps aux | grep java | awk '{print $2}' | xargs kill
```

Slide 11

Minute Essay

- Write a command to find all the files in the current directory (and subdirectories) that are less than a week old and list them in reverse order by modification time (i.e., newest to oldest).

Slide 12

Minute Essay Answer

- The solution I had in mind was

  ```
  find . -mtime -7 | xargs ls -lt
  ```

  but there are undoubtedly other ways!