Slide 1

Administrivia

- Reminder: Homework 1 due Wednesday at 5pm. Hardcopy please.

Slide 2

Minute Essay From Last Lecture

- “Surprising how many options a simple command (date) can have!”
- Lots of ways to terminate processes.

  (`kill` versus `kill -9`)

Slide 3

Pipes and Filters, Recap/Revisited

- Pipes allow you to connect output of one program to input of another. (There are also "named pipes" that work similarly and are persistent as opposed to single-use.)
- They're particularly attractive when combined with "filter" programs — and UNIX has lots of them, some of which seem kind of silly except for how well they work as building blocks.

Slide 4

Some 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.
- `cat` (concatenate — one or more inputs to output).
- `tee` — duplicates input. Good for capturing output to a file while also displaying it onscreen.

Slide 5

Examples

- Find all processes that belong to your username:

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

- Find all users who are running processes on the system:

  ```
  ps aux | awk '{ print $1 }' | sort | uniq
  ```

- Generate a list of machines that are "up":

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

  (Unfortunately this omits some machines, such as the dias cluster — different subnetwork.)

Slide 6

More Filters — `sed`

- `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 on next slide, with command inline. For more complicated edits, can put command(s) in a file.

Slide 7

Simple Examples of `sed`

- 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?) (`grep -v`!)

Slide 8

More Filters — `awk`

- `awk` — implementation of programming language AWK — "pattern scanning and processing language" (named after its inventors — as mentioned in its `man` page).
- 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 on next slide, with command inline. As with `sed`, for more complicated edits, can put command(s) in a file.

Simple Examples of awk

- 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?) (`grep`)

Slide 9

Still More Filters, and Other Useful Commands

- `diff` — compare files or directories. (A good use — “regression testing” of programs.)
- `xargs` — “build and execute command lines from standard input”.

  My standard(?) silly(?) example of the power of the command line:

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

Slide 10

Slide 11

Still More Useful Commands — `find`

- Very powerful/flexible, though there are so many options you probably won't remember them all. `man` page is useful if daunting! 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*"
  ```

  (Double quotes are needed so shell doesn't try to expand wildcard.)

Slide 12

`find`, A Bit More

- Summarizing and simplifying a bit from the `man` page, arguments to `find` consist of paths, "options", "tests", "actions", and "operators".
- Path(s) come first — where you want to search.
- "options" are next and apply to whole command, e.g. `-maxdepth`.
- Then there are "tests" (search criteria), "actions" (what you want to do with files that match — default is to print name), and "operators" (such as logical and and or) connecting them. Examples on next slides . . .

Slide 13

Examples of `find`

- Find all files in the current directory and subdirectories that end in `.bak` and remove them.

  ```
  find . -name "*.bak" -exec rm {} \;
  ```

  Here, `-name` is a "test" and `exec` an "action".

- As above, but prompt before executing each `rm`:

  ```
  find . -name "*.bak" -ok rm {} \;
  ```

  Here the "action" is `-ok`. (Might seem like you should be able to just use `rm -i`, but that doesn't work.)

Slide 14

More Examples of `find`

- Find files modified in last 24 hours and sort by modification time:

  ```
  find . -mtime -1 -type f | xargs ls -lt
  ```

  Here there are two "tests" (for time and type) and the default "action" (print) and we pipe into `xargs`

- But the above also lists files in `.cache`, which we may not care about. To exclude them, and also those in `mozilla` (should go all on one line):

  ```
  find . -name .cache -prune
  -o -name .mozilla -prune
  -o -mtime -1 | less
  ```

  This has three test-plus-action clauses, connected by `-o` (logical or) — two to tell `find` not to descend into directories we don't want, plus one that does what we want to the remaining files.

Slide 15

Minute Essay

- What command line could you use to count the number of aliases in your `.bashrc` file?

Slide 16

Minute Essay Answer

- One possible answer:

```
grep alias .bashrc | wc -l
```