Slide 1

Administrivia

- Reminder: I want a (short) project proposal from each of you, even if we discussed your project in person.

Slide 2

Installing and Updating Software — Package Managers

- "Modern" way to package software for installation depends on "package manager" — something that keeps track of what's installed, what depends on what, etc. (Examples — Fedora has `yum`, Debian has `apt-get`.) Software packaged as, e.g., `.rpm` or `.deb` files.
- If installing in "normal" system directories, and as root, probably best to take this approach.
- If you want to install in other directories (e.g., your home directory), or you don't have root access, some packages allow that, or you can (probably?) unpackage it Or there's the traditional UNIX approach . . .

Slide 3

Installing and Updating Software — "Tarballs"

- Traditionally, UNIX software distributed in the form of a "tarball" (archive created by `tar`, possibly compressed, usually containing source). Still often available and useful — e.g., to install in your home directory.
- What do you do with a tarball? Typical installation goes like this . . .

Slide 4

Installing and Updating Software — Installation from "Tarball"

- "Untar" the file (`tar xf`). Usually creates a directory, often containing `README` and/or `INSTALL` files — which you should review.
- Run `configure` script to set system-specific options. Usually figures most things out for itself, but may need/allow user input, either via command-line options or standard input.
- Run `make` to compile, etc. Normally puts created files in the same directory.
- Run `make install` to move/copy executables, etc., to system directories. Notice that this is the only step that requires root privileges — and only if installing in system directories.

Slide 5

More About Files — Permissions

- We talked about most of bits settable with `chmod` — r/w/x for u/g/o. But there are three more bits, applicable to directories and executable files . . .
- `setuid` "sets process's effective user ID". No effect on directories.
- `setgid` "sets effective group ID". Different effects on executables and (on some systems) directories.
- ``sticky'' . . . For executables, no longer used on some systems for original purpose. For directories, on some systems used for "restricted deletion".

Slide 6

More About Files — Hard Links Versus Soft Links

- Some background: UNIX filesystems traditionally keep track of files using "inodes". (I.e., directory entries point to inodes, which contain permissions, etc., plus info about file blocks.)
- "Hard link" points to inode.
- "Soft link" is just a file containing a path name.
- Which can point to a file on another filesystem? Which can be "broken"?

Slide 7

More About Processes and Shells

- You write a script to change directories. When you run it, what happens? Why?

  (Does this also explain why `man cd` gives you the `man` page for a shell?)

- What's the difference between "executing" a shell script and "sourcing" it?

Slide 8

Miscellaneous Useful Tips

- Recall many things you can do to replay commands from command history.
- Recall `pushd` and `popd`.
- If you know a lot of editor tricks, but only a few shell tricks, consider using editor to build temporary scripts. (Example(s).)

Slide 9

Some Questions from Minute Essays

- "How do you program a text-based game like the Tetris shown in class?"
  `ncurses` library provides text-only GUI-like functionality. Used by many programs that need this — `mutt`, etc. (See "Useful links" page here).
  (Another useful library — `readline`, which provies tab completion, command history, etc. `man readline` for more information.)
- "Any way to chat / send messages over network, like netsend in DOS"?
  There's `talk` — two-person text-based chat program, apparently blocked on lab machines.

Slide 10

Minute Essay

- None — sign in.