Outline

1. Shell Commands
2. Fabric
3. Demo Day
The subprocess module allows execution of shell commands

- `subprocess.call('ls')`

The commands are run in a child process

Longer commands can be specified with a list of strings

- `call(['grep', '-ir', 'python', './'])`

The I/O of the subprocesses can be set with kwargs

- `call('ls', stdin=f_handle, stdout=DEVNULL)`
Alternatives to Call

- `subprocess.call`
  - Executes the command
  - Waits until it exits
  - Returns the exit code

- `subprocess.check_call`
  - Just like `call` but ...
  - If the exit code is not one (Abnormal exit)
    - raise a `CalledProcessError`

- `subprocess.check_output`
  - Just like `check_call` but ...
  - returns the contents of stdout after the process finishes

- `subprocess.Popen`
  - Takes the same arguments as `call` (Except timeout)
  - Doesn’t wait for the process to finish
  - Have to check for output explicitly
  - More flexible for more complicated tasks
Shell globbing allows selection of files according to a limited regex
- * matches 0 or more of any character
- ? matches 1 of any character
- [a–z] character ranges in brackets
- /** matches any number of directories

A convenient way to select files
Essentially walks the directory applying a modified regex

```python
from glob import glob, iglob

glob(path_regex) returns a list of matching files
iglob(path_regex) an iterator version of glob
```
Using `subprocess` you can pipe the output of one process to the input of another process.

The easiest way: Just use `' | '` in a subprocess command.

```
subprocess.check_call(['ls', ' | ', 'wc', ' -l'])
```

`subprocess.PIPE` lets you do this with more control.

Must use `subprocess.Popen` instead of `subprocess.call`.

```
subprocess.call waits for the process to finish
```

If the pipe fills up then the process will deadlock.
Outline

1. Shell Commands
2. Fabric
3. Demo Day
Installing Fabric

```
pip install fabric
```
Fabric is “a Python library and command-line tool for streamlining the use of SSH for application deployment or systems administration tasks.”

Fabric has two main features:
1. Easy use of the terminal from within a python program.
2. Allows multiple computers to easily communicate and coordinate tasks.
$ fab func will call the func method in fabfile.py
  ▶ Optional arguments go after the colon.
  ▶ eg. $ fab hello:name=Bob will call hello with the argument name set to “Bob”.

If the file isn’t called fabfile.py use
$ fab -f [filename] [function_name]
There are multiple ways of specifying the remote host(s).

1. `fab -H system1,system2,...` specifies the system(s) upon call.
2. `env.hosts = system1,system2,...` in the fabric file.
3. The decorators `@hosts(system1,system2,...)` can be applied to individual functions.
4. Otherwise, fabric will ask for the system upon execution.
local(command) runs a command on the local machine.
  ▶ Set capture = True to return the output instead of sending it to stdout.
run(command) runs a command on the remote machine.
  ▶ Set stdin = x and stdin = y to pass stdin and stderr to the variables x and y.
Get and Put

Both `get` and `put` take in a local_path and remote_path, relative to the current local and remote directories. (Absolute paths are also permitted.)

- `put` transfers a file from the local machine to the remote machine.
- `get` does the reverse.
- Use `with cd(dir)` to run all commands from a given directory.
Outline

1. Shell Commands
2. Fabric
3. Demo Day
Demo Day

- **Wednesday, December 9th at 12:00pm**
- All students are expected to give a working demo of their projects.
- If you like, this can be your final demo of the project (no need to demo again on December 18th). Email me beforehand if that is your preference.
- If you cannot make the Demo Day, email me.
- There will be prizes and food and judges from various tech companies.