A different take on more scripting stuff;& version control

LECTURE 13
OCT 18, 2010

But first…

Whither the LTEE project?

- Many of you are still working out the details with Heather (and occasionally me)
- Those that aren't should try running it.
- What do you all want the output to be?
  - Report?
  - 2-5 minute presentation?
  - A general feeling of satisfaction?
- And when?

Class #2

The Ian and Charles show

- All BEACON grads, anywhere, must take the second class.
- (New) groups will be formed across discipline and institution boundaries
- View this class as a warm-up

Back to this class

- Where do you want *this* class to go?
- More practice?
- More Cool Science?
- More on programming?

Version control

- Think of it as Word's "change tracking" on text files.

- Many different version control "systems" –
- Centralized:
  - **Subversion**
  - CVS

- Distributed:
  - **git**
  - Mercurial
  - Arch, Bzr, Darcs, …

Version control vocabulary

- Repository – location of all the files
- Working directory – copy of repository where you "do work" (make changes, use code, etc.)
  - Linked to the main repository
  - Created via a "checkout" or a "clone" command.
  - "Push" changes *to* the repository and "pull" changes *from* the repository.
  - Similar to "commit" (push) and "update" (pull) in Subversion

Github is a hosting service for git repositories

- I'm hosting the BEACON class scripts on github.
- This means that *all of you* can make a checked-out copy of the files.
- You can also pull updates. More on that in a bit.
- You can *also* sign up for github and make your own changes.

Other reasons to use version control

- It keeps a complete history of changes to the source code, so you can always go back and figure out when a feature was removed or a bug was introduced.
- You can “checkpoint” your source code for later reference (“this was the version I used for paper XXX”)
- You can “branch” your source code to work on independent features/scripts/etc., and then later on “merge”.
- It’s essential to multi-developer projects.

Other reasons to use version control, #2

- It's a backup! And with github and other hosting services, it's a *free* backup! (But your code does have to be open source…)

- It's a pretty convenient way to transfer files around.

Version control is critical to replication

- Being able to say “this code is identical to the code I ran a year ago” is really important! (Or, “this is the version I ran a year ago”)
- You can make that source code available to *others* quite easily via a version control system.
- It’s also nice to be able to see “hey, feature XXX isn’t working any more, and person YYY *is the person who broke it*.”

Version control isn't a panacea!

- Just because it's in version control doesn't mean it's right.
- Just making code available to someone doesn't mean they can actually run it.
- Many people don't include *all* their code.

beacon/ltee/run-next-2.py

```
1  # run-next-2: run a serial transfer experiment, always transferring N
2  #    random organisms
3
4  N = 10
5
6  assert N >= 1, "invalid N! must be positive."
7
```

Parameters at top!

Make sure that the parameters are not stupid!

beacon/ltee/run-next-2.py

```
16  # retrieve parent directory from command line
17  parent_dir = sys.argv[1]
18  run_dirs = os.path.join(parent_dir, 'run.*')
19  run_dirs = glob.glob(run_dirs)
20
21  # find the last one -- not alphabetic last, but numeric last!
22  run_nums = []
23  for dir in run_dirs:
24      num = dir.split('.')[-1]
25      print num
26      try:
27          num = int(num)
28      except ValueError:
29          continue
30
31      run_nums.append(num)
32
33  last_run = max(run_nums)
34
35  last_run_dir = 'run.%d' % last_run
36  last_run_dir = os.path.join(parent_dir, last_run_dir)
```

Load in the existing run dirs

beacon/ltee/run-next-2.py

```
40  # ok, we've got the last one.  make a new one!
41  new_run_dir = 'run.%d' % (last_run + 1)
42  new_run_dir = os.path.join(parent_dir, new_run_dir)
43
44  template_dir = os.path.join(parent_dir, 'run.template')
45
46  shutil.copytree(template_dir, new_run_dir)
47
```

Make a new directory and copy the template to it.

beacon/ltee/run-next-2.py

```
48  #
49  # use 'dominant.dat' to figure out what the last update was.
50  #
51
52  last_run_data = os.path.join(parent_dir, last_run_dir, 'data')
53  dominant_file = os.path.join(last_run_data, 'dominant.dat')
54  fp = open(dominant_file)
55
56  last_line = None
57  for line in fp:
58      line = line.strip()
59      if line:
60          last_line = line
61
62  last_line = last_line.split()
63
64  last_update = last_line[0]
65
```

Use ‘dominant.dat’ to get the last update.

beacon/ltee/run-next-2.py

```
66  ### now get the actual genomes from the 'detail-' file:
67
68  population_file = 'detail-' + last_update + '.spop'
69  population_file = os.path.join(last_run_data, population_file)
70
71  all_organisms = []
72
73  fp = open(population_file)
74  for line in fp:
75      line = line.strip()
76      if not line or line.startswith('#'):
77          continue
78
79      line = line.split(' ')
80      organism = line[0]
81      genome = line[16]
82      all_organisms.append((organism, genome))
83
```

Load the genomes into ‘all_organisms’ list.

beacon/ltee/run-next-2.py

```
84  # choose N, randomly.
85  transfer_pop = random.sample(all_organisms, N)
```

Self-explanatory!

beacon/ltee/run-next-2.py

```
87  # ok, now we have to do two things: first, we have to stick in the *starting*
88  # organism, which we'll make the first of the critters we selected.  All others
89  # will be inserted using 'InjectSequence'.
90
91  # get first organism.
92  (org_id, first_genome) = transfer_pop[0]
93  print transfer_pop[0]
```

Seed Avida with the first organism.

```
95  # OK!  now translate.
96  instset_filename = os.path.join(new_run_dir, 'instset-heads.cfg')
97  instructions = open(instset_filename)
98
99  # build the dictionary mapping instruction characters to instructions
100 char_to_inst = {}
101 alphabet = 'abcdefghijklmnopqrstuvwxyz'
102
103 n = 0
104 for line in instructions:
105     line = line.strip()
106
107     if not line:
108         continue
109     if line.startswith('#'):
110         continue
111
112     # all right, non-empty line... save instruction with corresp character
113     the_inst = line.split()[0]
114     char = alphabet[n]
115     char_to_inst[char] = the_inst
116     n += 1
117
```

Load in the instruction translation table:

a => nop-A
b => nop-B
w => h-alloc
etc

```
120  org_name = 'run.%d-first.org' % last_run
121  org_filename = os.path.join(new_run_dir, org_name)
122  orgfp = open(org_filename, 'w')
123
124  print >>orgfp, "# organism %s from run %d" % (org_id, last_run)
125  for ch in first_genome:
126      print >>orgfp, char_to_inst[ch]
127  orgfp.close()
128
```

Here, we're outputting the translation of 'first_genome' into the file 'run.N-first.org'

Run-next-2.py

```
131 old_cfg = os.path.join(new_run_dir, 'avida.cfg.bak')
132 new_cfg = os.path.join(new_run_dir, 'avida.cfg')
133 shutil.move(new_cfg, old_cfg)
134
135 outfp = open(new_cfg, 'w')
136 for line in open(old_cfg):
137     if line.startswith('START_CREATURE'):
138         print >>outfp, 'START_CREATURE', org_name
139     else:
140         outfp.write(line)
141 outfp.close()
```

- Update the config file

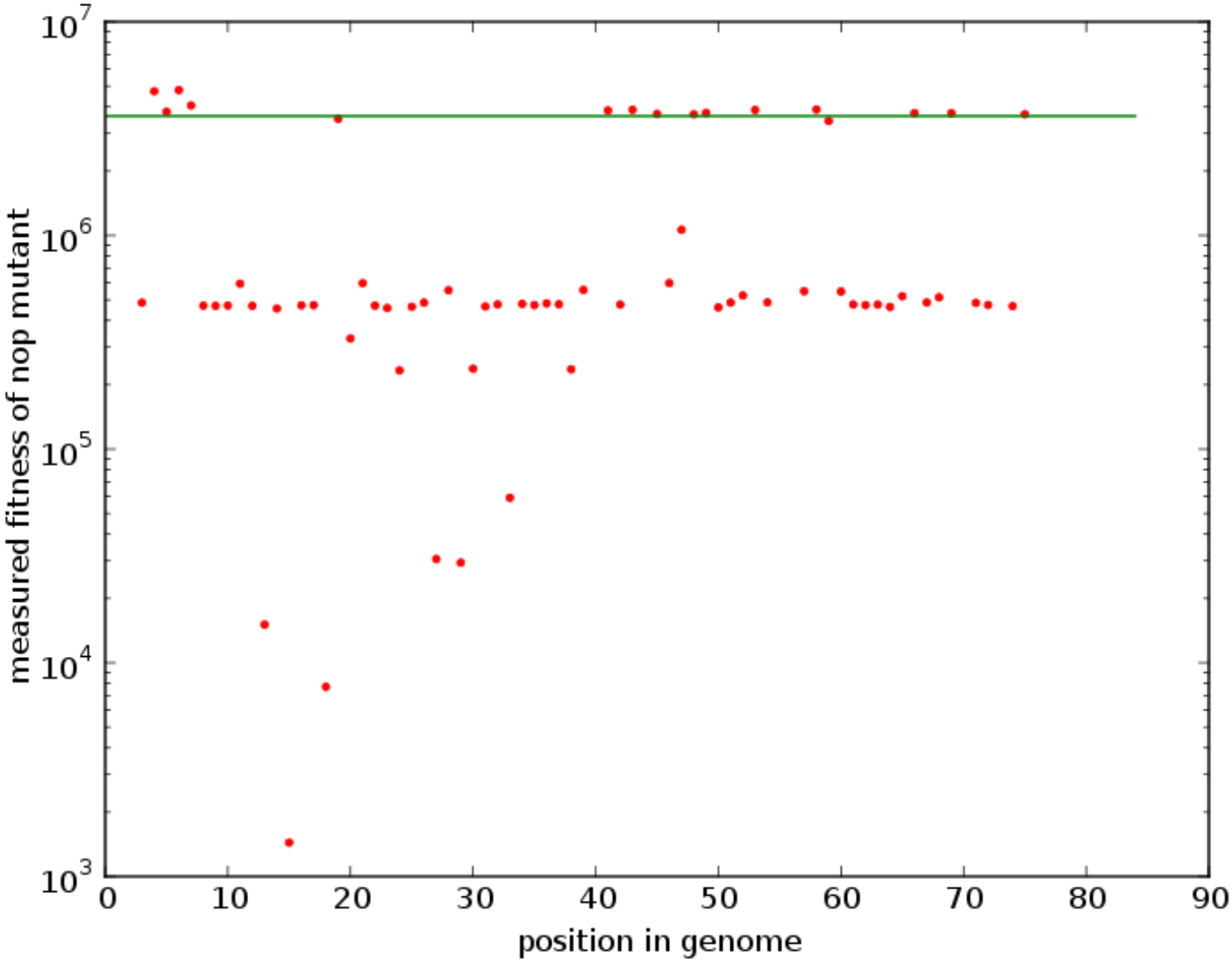
Run-next-2.py

```
145  # now, also modify the events file to inject the remaining critters
146  eventsfp = open('events.cfg', 'a')
147
148  print >>eventsfp, "\n# injecting as part of serial transfer:"
149  for (org_id, genome) in transfer_pop[1:]:
150      print >>eventsfp, "u 0 InjectSequence %s  # organism %s from run %d" % \
151          (genome, org_id, last_run)
152
153  ###
154
```

- Inject the rest!

Simple version control commands

git clone <git repository>	- make a working copy
git pull <git repository>	- update from repo
git log	- see log of changes
git diff <revision ID>	- see ‘diff’


10^7
10^6
10^5
10^4
10^3
measured fitness of nop mutant
0
10
20
30
40
50
60
70
80
90
position in genome