Python & Git Basics

Intro to git

git is a "third generation" source control system

source control is used for:

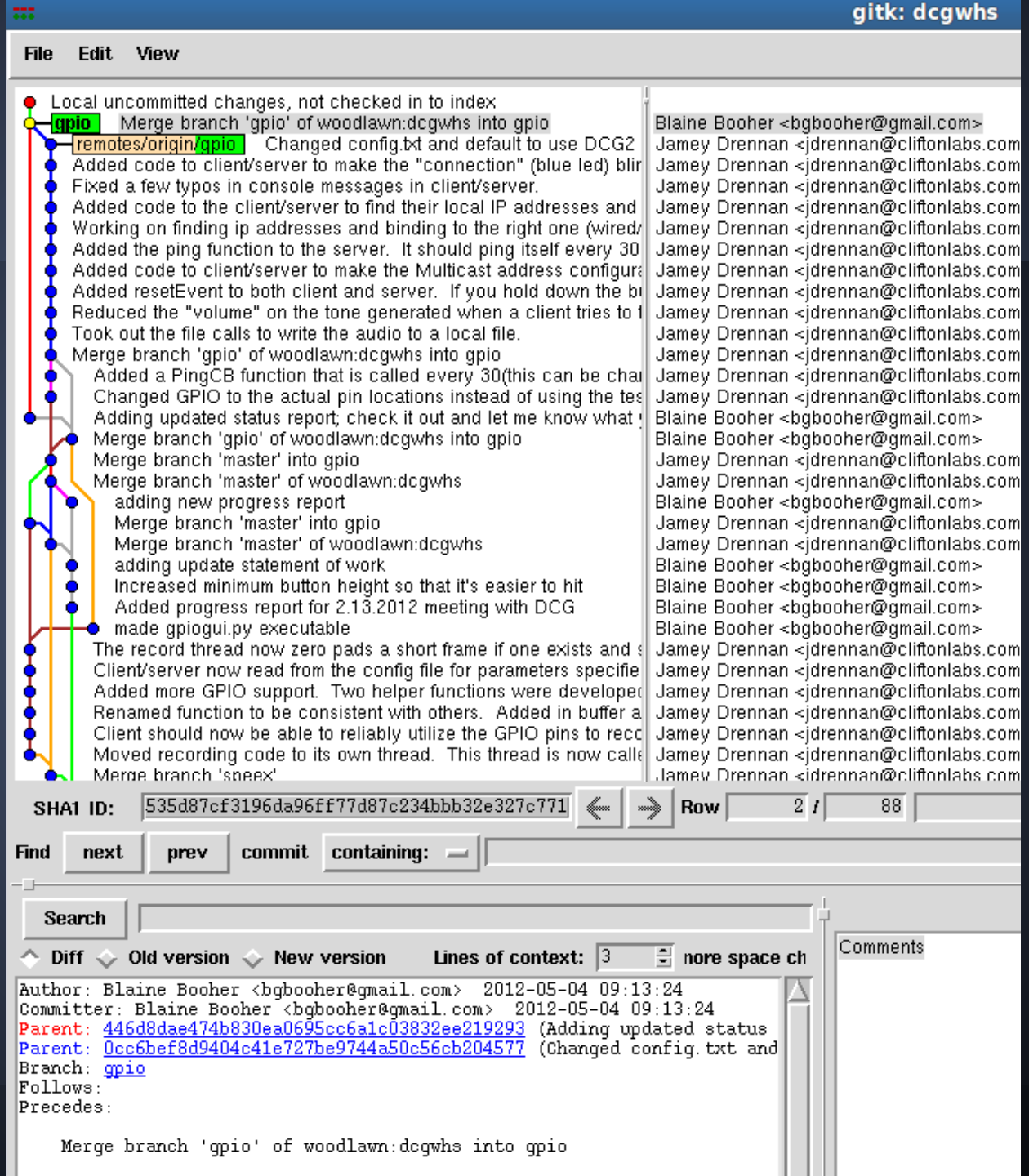
- Keeping track of changes
- Allows going back to old versions
- Track developers & contributions
- Easy collaboration with branching
- Distributed development across networks

git

git was created by Linus Torvalds in, like, a weekend. Seriously.

- It is very fast, written in pure C
- You don't have to be online to make changes, unlike SVN. This will make sense soon.
- Lots of tools for visualizing git
  - gitk (visualize branches and history)
  - git-cola (GUI for checkins, staging, etc.)

gitk


gitk: dcgwhs
File Edit View
Local uncommitted changes, not checked in to index
gpio Merge branch 'gpio' of woodlawn:dcgwhs into gpio
remotes/origin/gpio Changed config.txt and default to use DCG2
Added code to client/server to make the "connection" (blue led) blir
Fixed a few typos in console messages in client/server.
Added code to the client/server to find their local IP addresses and
Working on finding ip addresses and binding to the right one (wired/
Added the ping function to the server. It should ping itself every 30
Added code to client/server to make the Multicast address configura
Added resetEvent to both client and server. If you hold down the bu
Reduced the "volume" on the tone generated when a client tries to f
Took out the file calls to write the audio to a local file.
Merge branch 'gpio' of woodlawn:dcgwhs into gpio
Added a PingCB function that is called every 30(this can be chai
Changed GPIO to the actual pin locations instead of using the tes
Adding updated status report; check it out and let me know what
Merge branch 'gpio' of woodlawn:dcgwhs into gpio
Merge branch 'master' into gpio
Merge branch 'master' of woodlawn:dcgwhs
adding new progress report
Merge branch 'master' into gpio
Merge branch 'master' of woodlawn:dcgwhs
adding update statement of work
Increased minimum button height so that it's easier to hit
Added progress report for 2.13.2012 meeting with DCG
made gpiogui.py executable
The record thread now zero pads a short frame if one exists and s
Client/server now read from the config file for parameters specifie
Added more GPIO support. Two helper functions were developed
Renamed function to be consistent with others. Added in buffer a
Client should now be able to reliably utilize the GPIO pins to recc
Moved recording code to its own thread. This thread is now calle
Merge branch 'sneex'
Blaine Booher <bgbooher@gmail.com>
Jamey Drennan <jdrennan@cliftonlabs.com
Blaine Booher <bgbooher@gmail.com>
SHA1 ID: 535d87cf3196da96ff77d87c234bbb32e327c771
Row 2 / 88
Find next prev commit containing:
Search
Diff Old version New version Lines of context: 3 nore space ch
Comments
Author: Blaine Booher <bgbooher@gmail.com> 2012-05-04 09:13:24
Committer: Blaine Booher <bgbooher@gmail.com> 2012-05-04 09:13:24
Parent: 446d8dae474b830ea0695cc6a1c03832ee219293 (Adding updated status
Parent: 0cc6bef8d9404c41e727be9744a50c56cb204577 (Changed config.txt and
Branch: gpio
Follows:
Precedes:
Merge branch 'gpio' of woodlawn:dcgwhs into gpio

git basic

git clone: allows you to pull down a new repository
git pull: pulls down latest changes and merges them into your branch
==> 1. Fetch 2. Merge
When you modify a file and want to update it:
git add myfile.py (this stages the file)
git commit -m "Added a new class to myfile.py " (always leave good notes)
==> So at this point it is now committed to your LOCAL repository.
==> See history with: git log
if you have an upstream repo defined, we can see that here:
76$ git remote -v
origin git@woodlawn:dcgwhs (fetch)
origin git@woodlawn:dcgwhs (push)
Finally, to push them to your remote (usually origin is default, master branch)
git push origin master (or just git push)

summary

Other useful commands:

git branch mybranch (creates a new branch to work on in parallel)

git merge mybranch (merges the mybranch branch into your working branch)

git branch (shows all branches on your repo)

git checkout mybranch (switch branches)

git log (shows git log messages)

git diff myfile.py (shows diff between repo and local version of myfile.py)

git checkout <hash> (check out an old version of the repo. every checkin and file has a hash associated with it. See git log for hash information.

example hash: 0cc6bef8d9404c41e727be9744a50c56cb204577

Python Classes

A python class is so simple. It's beautiful.

```
class myclass(object):
    def __init__(self, name):
        ''' Initialization Routine '''
        self.name=name
        print "my name is %s" % name

    def reverse(self):
        ''' A simple helper function '''
        x = self.name
        x.reverse()
        print "my reversed name is: %s" % x
```

More information

class myclass(object):

The python object 'object' is the root object that everything inherits from. class myclass() works too, but it's cleaner to always include object.

To inherit another class:
class myclass(XMLWriter):

Classes, continuing

```
def __init__(self, name):
   self.name = name
```

the __init__ is a special name that represents the constructor of the python class.

self is a pointer to the class itself and *must* be present as the first parameter in every instance function. name, of course, is the only real parameter.

we set the class variable (for this instance) by referencing 'self'.

We initialize our class this way:

```
m = myclass('blaine')
```

```
    def reverse(self):
        ''' A simple helper function '''
        x = self.name
        x.reverse()
        print "my reversed name is: %s" % x
```

A simple class routine that accesses the 'self' variable to grab our instance variable 'name'. We do a simple reverse on it and print the value. We could just as easily return the value, too.

```
m = myclass('blaine')
# output: my name is blaine
m.reverse()
# output: my reversed name is enialb
```

Other special class stuff

__init__(self) is just one special function

__iter__, __next__ are for working with iterators

__call__(), if present, allows you to call an object.

```
    x = myclass('name')
    x()   # this is calling the class as if it were a function
```

__repr__ and __str__ allow you to get a human readable name of the class, great for printing

__len__ is meant to return the "length" of the object, useful if you're working with a list object of somekind

__getitem__(x) returns item x. example:

```
class myclass(object):
    def __init__(self)
      self.x = (0,1,2,3,4,5,6,7,8,9,10)
    def __getitem__(self, i)
      return self.x[i]
m = myclass()
print m[5]   # prints 4
```

exceptions

Try / Except clauses are easy. This will catch every single exception thrown:

```
try:
   do_some_stuff
except:
   print "error"
```

exceptions...

One useful exception is for working with dictionaries

```
x = {'name': 'blaine', 'location': 'cincinnati'}

print x['name']
# prints blaine

x['age'] # throws KeyError Exception
In [17]: x['age']
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
KeyError: 'age'
```

exceptions

two easy ways to solve this problem:

```
if x.has_key('age'):
   print x['age']
else:
   print 'sorry key not found'

try:
   print x['age']
except KeyError:
   print 'sorry key not found'
```

Sometimes it's just easier to do everything in except clauses. when a new exception pops up, you can just account for it and move on without letting the program crash

importing modules, exceptions

```
numpy = None
try:
   import numpy
except ImportError:
   print "Can't import numpy, no matrix support"

# then, later:
if numpy:
   n = numpy.lots_of_matrix_math()
else:
   n = my_slow_matrix_math()
```

Map

Let's say you want to take a list of strings and convert them to integers. One way to do this is by using map(FUNC, LIST). Map simply says "take every item in the LIST and run them through FUNC, storing the results in a table.

l = ('4', '6', '3')

map(int, l)

output: [4,6,3]

We use 'int' here just because we can call int(k) where k is a string. but what if we need it to do something else? You can either make a dedicated function for every different usage of map(), or use lambda functions.

Lambda Functions

Lambda functions are executable objects that are created in-place without a name. They can be passed around like functions (because they are functions!). They're slow but convenient depending on the usage.

Here's a lambda function that takes a single parameter called 'o', and returns that parameter times 5:

lambda o: o*5

Here's a lambda function that takes NO parameters, and just returns 5!

lambda: 5

Lambda Functions (cont)

You can assign the lambda objects to a variable and use them like a normal function!

```
x = lambda o: o*5
y = lambda: 5
>>> x = lambda o: o*5
>>> y = lambda: 5
>>> x('a')
'aaaaa'
>>> x(3)
15
>>> y()
5
>>> x(y())
25
```

Here we also see examples of python's runtime flexibility. As long as the parameter passed into x supports the operator (like a str and int do), the lambda function has no problem with returning o*5

Lambda + Map

Now we start to see where lambda can be useful with map.

```
l = ('3', '4', '5')
map(int, l) # this is easy because we already know int takes one param
```

But what if we want to get the second part of each element to turn into an int?

We could create a new function or use a lambda function

```
def convert(o):
   i,k = o.split(",")
   return k
l = ('a,4', 'b,5', 's,8')
map(convert, l)   # Use our existing function
map(lambda o: o.split(',')[1], l)  # Create a lambda function, and use that object
```

This works because all map cares about is that the first argument is a callable object. That object can be a real function, a class instance with __callable__ defined, or a lambda function.

[o for o in range(10)]

This is basically an "in line" for-loop.

One way I like to use it is for reading files and parsing in one tight line.
Imagine we have a .csv file like this:

1, 2, 5, 6
5, 4, 3, 5
1, 2, 3, 3

This will read each line, split by ",", and use map() to convert the strings to integers using a lambda function that first strips the object of newlines.

```
[map(lambda o: int(o.strip()), i.split(',')) for i in open("file.csv").readlines()]
```

Note: This one is kind of a beast. Sometimes it is easier for readability to break it all out into individual lines. It's up to you.

easy_install, pip, virtualenv

easy_install and pip are two great programs for installing python modules. Both are available in aptitutde or apt-get with ubuntu.

I use them both interchangeably. I'm starting to prefer pip due to its support for virtualenv.

sudo easy_install package_name
sudo pip install package_name

Working with setup.py

If you download a .zip file for a python package and aren't sure what to do, it's actually quite simple. Always look for setup.py

You may need the package python-setuptools for all variants of setup.py.

```
$ tar -zxvf mymodule.tar.gz
$ cd mymodule/
$ python setup.py install
OR
$ python setup.py build_ext        # means build the python extension
$ sudo python setup.py install     # once built, actually install it but needs root permissions
```

For larger packages, this can take a long time. Behind the scenes python is calling gcc to compile stuff, if necessary, and grabbing all the necessary packages etc.

virtualenv

We'll cover more about virtualenv later, but for now we should be aware of its existence.

Virtualenv allows us to create a "virtual environment" in which we work with python projects. We can install, using pip, all kinds of packages to this virtual environment and they are not accessible outside of this virtual environment to the base system.

This is excellent for managing web projects because you can keep packages with different versions on the machine without conflicting.

You can also deploy easily because pip is aware of which packages it has installed to the virtualenv. "pip freeze" will list all packages installed.

ex: excel data reading (xlrd module)

```
[blaine@macbook:/data/cliftonlabs/ken_hill Fri May 04]
85$ sudo easy_install xlrd
[sudo] password for blaine:
Searching for xlrd
Reading http://pypi.python.org/simple/xlrd/
Reading http://www.lexicon.net/sjmachin/xlrd.htm
Best match: xlrd 0.7.7
Downloading http://pypi.python.org/packages/source/x/xlrd/xlrd-0.7.7.tar.gz#md5=ba46c586b4c1df977bf7bdcd711590bf
Processing xlrd-0.7.7.tar.gz
Running xlrd-0.7.7/setup.py -q bdist_egg --dist-dir /tmp/easy_install-jZvt6u/xlrd-0.7.7/egg-dist-tmp-Fz6Fu5
zip_safe flag not set; analyzing archive contents...
Adding xlrd 0.7.7 to easy-install.pth file
Installing runxlrd.py script to /usr/local/bin

Installed /usr/local/lib/python2.6/dist-packages/xlrd-0.7.7-py2.6.egg
Processing dependencies for xlrd
Finished processing dependencies for xlrd
```

Excel Spreadsheet Reading

Here's a great list of data sets:

http://mathforum.org/workshops/sum96/data.collections/datalibrary/data.set6.html

See if you can install the module and use it to read data from one of those spreadsheets.

For bonus, use the "urllib" library's urlopen (url).read() routine to save the file to disk before processing it.