Outline:

- A few words on coding style
- A few words on packages in Python
- Managing development with version control systems


Every time
you don't
use version
control, I kill
a kitten.

Further summary points:

- The best VCS is “git”
- Invest some time and concentration now,

  reap rewards soon and forever.
- “PEP8” gives standard Python coding style
- Don't just write code – read it too

Coding Style!

- PEP8: http://python.org/dev/peps/pep-0008/

```
def frob_widgets(x, y, z, kw=0):
    # set y to 1 if z is zero
    if z == 0: y = 1
    # Might need to account for margin pixel
    if z == 0:
        y = 1
    subfunc(y, z + 1, x[0], {'magic': True})

class WidgetFrobber(object):
   ...

import telemetry
```

- Don't forget to read code as well as write it! (*E.g.*: the huge Python standard library)

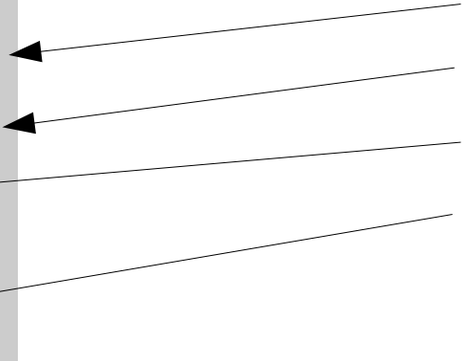
Packages!

For when it's getting a little silly to cram everything into one module.

```
myprogram.py
scope/
scope/__init__.py
scope/pointing.py
scope/dome.py
scope/fans.py
myutils.py
```

myprogram.py:

```
import scope
import scope.pointing
import scope.dome
import myutils
```



setup.py / distutils

The standard Python system for distributing and installing packages

```
$ cd downloaded-package-1.2
$ ./setup.py build
[ ... ]
$ ./setup.py install # might need sudo or --prefix
[ ... ]
```

Non-Linux, Linux without root access:

```
$ easy_install pyfits # might need sudo or --prefix
```

Linux with root access: use your system package manager (else face nerd-wrath)

setup.py / distutils

```
from distutils.core import setup
from distutils.extension import Extension
import sys

def main():
    setup(name = "pysao",
          version = "0.1b1",
          description = "python wrapper around some SAO tools",
          author = "Jae-Joon Lee",
          author_email = "lee.j.joon@gmail.com",
          maintainer_email = "lee.j.joon@gmail.com",
          url = "",
          license = "MIT",
          platforms = ["Linux","Mac OS X"], # "Solaris"?
          packages = ['pysao'],
          package_dir={'pysao':'lib'},
          ext_modules=[ Extension("pysao.xpa",        ["xpa.c"],
                                  include_dirs=['./xpalib'],
                                  library_dirs=['./xpalib'],
                                  libraries=['xpa']),
                        ],
          )

if __name__ == "__main__":
    main()
```

Version control — what, why?

First-order understanding:

- Checkpoints in evolution of source code
- Allow multiple developers to collaborate on a single codebase

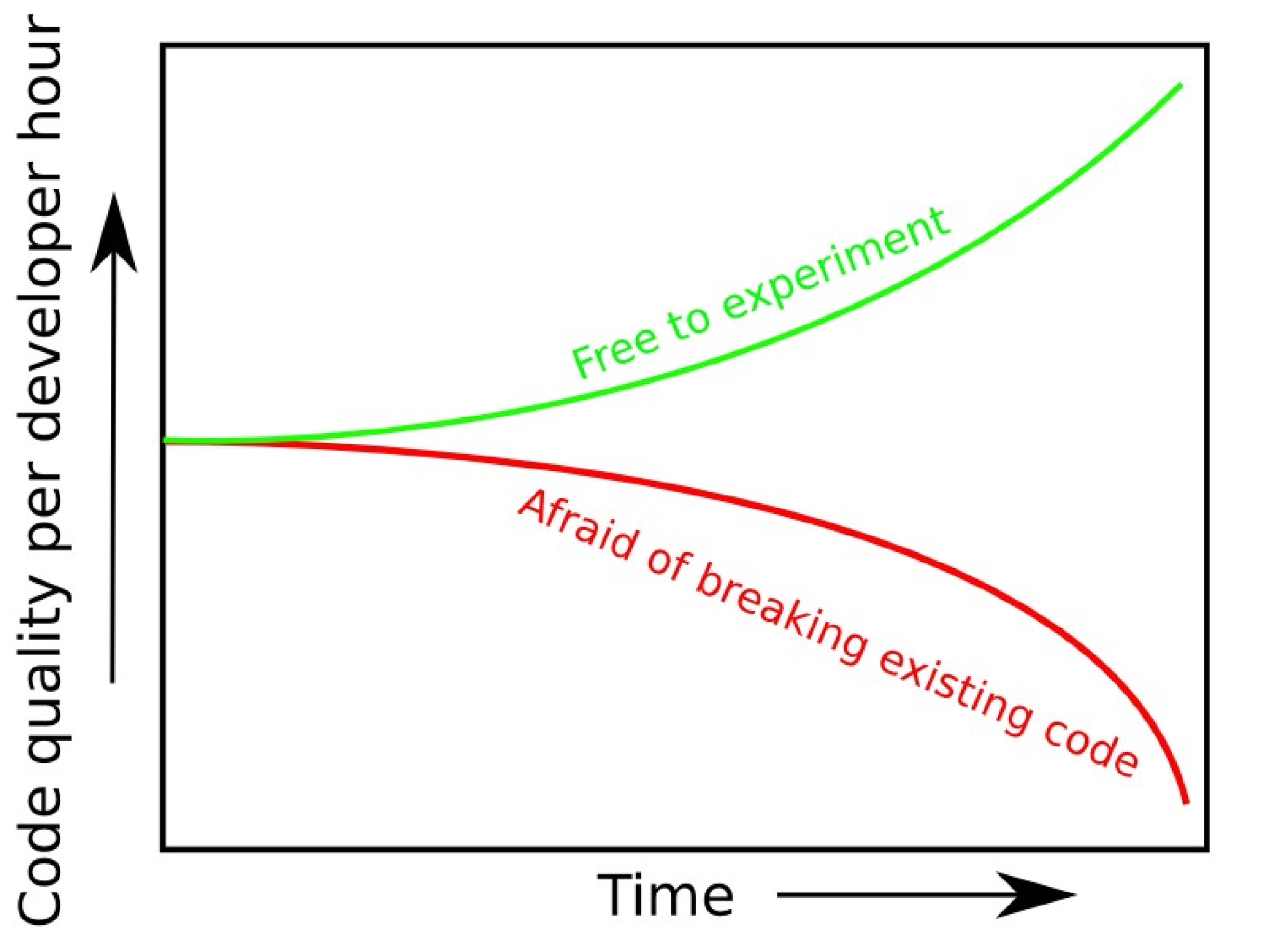
The Generic Version Control System

- Files and development history stored in “repository”
- “Check out” files into a working area
- “Commit” changes to files back to repo
- “Update” your working area with other developers' commits

Version control — what, why?

Deeper points:

- The project history becomes immensely valuable
- No need to live in fear


Code quality per developer hour
Free to experiment
Afraid of breaking existing code
Time

Google

git tutorial

About 1,880,000 results (0.13 seconds)

Everything
More

All results
Wonder wheel
Timeline
More search tools

gittutorial(7)

Jul 21, 2010 **... gittutorial** - A tutorial introduction to git (for version 1.5.1 or ne
tutorial explains how to import a new project into git, **...**
www.kernel.org/pub/software/scm/**git**/docs/**gittutorial**.html - Cached - Simila

A **tutorial** introduction to **git**

Feb 13, 2007 **...** Other good starting points include Everday GIT with 20 Co
and git for CVS users. Also, A short **git tutorial** gives an **...**
www.kernel.org/pub/software/scm/**git**/docs/v1.../**tutorial**.html - Cached - Si

Show more results from www.kernel.org

Understanding **Git** Conceptually

Apr 17, 2010 **...** This is a **tutorial** on the **Git** version control system. **Git** is qu
becoming one of the most popular version control systems in use. **...**
2: Branching - Merging - Rebasing - Repositories
www.eecs.harvard.edu/~cduan/technical/**git**/ - Cached - Similar

Git - SVN Crash Course

You will need the latest **Git** installed; There is also a potentially useful **tutoria**
documentation. How to Read Me; Things You Should Know **...**
git.or.cz/course/svn.html - Cached - Similar

Git: Getting Started

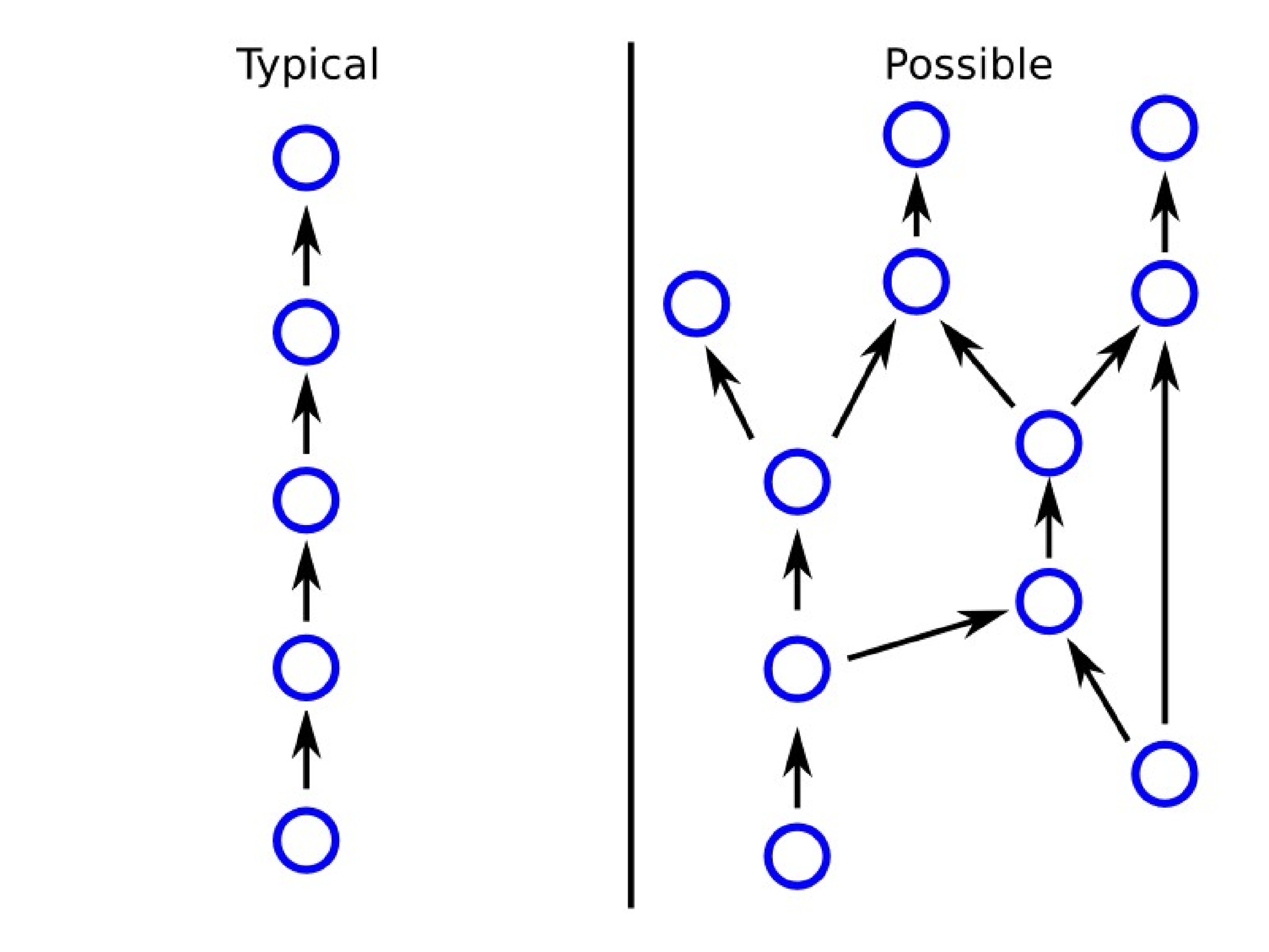
```
$ mkdir myawesomesoftware
$ cd myawesomesoftware
$ git init
Initialized empty Git repository in [path]
$ ls -a
. .. .git
$ emacs mystuff.py
$ git add mystuff.py
$ git commit -m “Initial import of my stuff.”
[master (root-commit) 1234abcd] Initial import...
  1 files changed, X insertions(+), 0 deletions(-)
  create mode 100644 mystuff.py

$ git help commit
[ ... ]
```

Git: Underlying model AKA sorry: math

- All git does is keep a database of “commits”
- Commit ≡ (tree, author, timestamp, log message, parent commit[s])
- (formally: “directed acyclic graph”)
- Named with magic numbers eg.

  `0d30e664c0839392a0ec8c7c266e9e194b8bb7f6`

- All you do is create new commits based off of old commits


Typical
Possible

Git: Underlying model AKA sorry: math

- Git makes this efficient. Don't worry.
- “Make the easy things easy and the hard things possible”

Git: Basic Changes

```
$ # what's the One True Name of the commit I'm on right now?
$ git rev-parse HEAD
e021e21ebbca312e8c76e618e30883a84d27b3ac

$ git status
# On branch master
nothing to commit (working directory clean)

$ emacs mystuff.py
$ git status
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in wo...
#
#       modified:   mystuff.py
#
no changes added to commit (use "git add" and/or "git commit -a")
```

YouTube break!!!1!!1!!!

Git: Basic Changes

```
$ # what's the One True Name of the commit I'm on right now?
$ git rev-parse HEAD
e021e21ebbca312e8c76e618e30883a84d27b3ac

$ git status
# On branch master
nothing to commit (working directory clean)

$ emacs mystuff.py
$ git status
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in wo...
#
#       modified:   mystuff.py
#
no changes added to commit (use "git add" and/or "git commit -a")
```

Very Important Aside: Diffs (AKA patches)

```
$ git diff
diff --git a/mystuff.py b/mystuff.py
index 93ec7d1..7e9b565
--- a/mystuff.py
+++ b/mystuff.py
@@ -3,7 +3,10 @@
 print 'Hello, world!'

 for i in xrange (10):
-    print i, '...'
+    print 10 - i, '...'
+
+print '... blastoff!'

 print 'Two plus two is', 2 + 2

+print 'Five times five is', 5 * 5
$ git diff >fix-peters-bug.diff
```

See also: `patch`, `diff -u`, `diffstat`, every other VCS ever, developer mailing lists for every open-source project ever

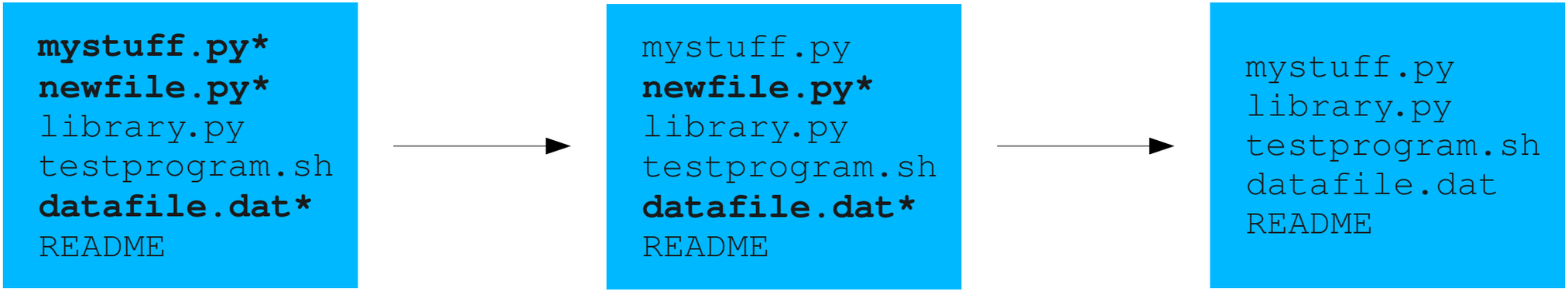
Before:

```
#! /usr/bin/env python

print 'Hello, world!'

for i in xrange (10):
    print i, '...'

print 'Two plus two is', 2 + 2
```

After:

```
#! /usr/bin/env python

print 'Hello, world!'

for i in xrange (10):
    print 10 - i, '...'

print '... blastoff!'

print 'Two plus two is', 2 + 2

print 'Five times five is', 5 * 5
```

Git: Creating commits
(only a few more formalities to work through ...)



“Working tree”
“checkout”

“Index”
“staging area”

“HEAD”

- Allows you to separate “getting code written”

  and “making nice commits”

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#    modified:   newfile
#
# Changed but not updated:
#   (use "git add <file>..." to update what will be
#    committed)
#   (use "git checkout -- <file>..." to discard changes
#    in working directory)
#
#    modified:   mystuff.py
#
# Untracked files:
#   (use "git add <file>..." to include in what will be
#    committed)
#
#    anotherfile
$
```

```
$ git diff
```

Diff between **working tree** and **index**

```
$ git diff --staged
```

Diff between **index** and **HEAD**

```
$ git diff <commit>
```

Diff between **any commit** and **HEAD**

```
$ git diff <commit1> <commit2>
```

Diff between **any two commits**

Git: Basic Changes

```
$ git status
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in wo...
#
#      modified:   mystuff.py
#
no changes added to commit (use "git add" and/or "git commit -a")
$ git diff
diff --git a/mystuff.py b/mystuff.py
index 93ec7d1..7e9b565
--- a/mystuff.py
+++ b/mystuff.py
@@ -3,7 +3,10 @@
 print 'Hello, world!'

 for i in xrange (10):
-    print i, '...'
+    print 10 - i, '...'
+
+print '... blastoff!'

 print 'Two plus two is', 2 + 2

+print 'Five times five is', 5 * 5
$ git commit -a -m "mystuff: reverse countdown, check multiplication"
```

Git: Basic Changes

- Commits are *fast* and *cheap* – do them often
- (Ideally....) Commit diffs should be clean, practically and conceptually
- Use the index to put together nice commits
- Look at diffs while you work to help you think about the changes you're making


Every time you write a bad commit message, I kill two kittens.

```
commit 30db5d9ef97f34d4de50f4c5a8beb46a11d2bd4d
Author: redacted
Date:   Wed Jul 7 15:23:44 2010 +0000

    new
```

```
commit a572adbef9138ade4cbd68d384b17c40b49a1e3b
Author: me
Date:   Tue Jul 6 20:34:25 2010 +0000

    configure.ac: move F77 test first for more thorough testing

    It appears that the first compiler tested by autoconf is the only one
    that has more thorough checks performed about its ability to
    produce executables. The Fortran compiler is the one that's most
    likely to give problems, so have it be the first one checked.
```

Git: Branches

Branch ≡ user-friendly name for a commit

```
$ git status
# On branch master
[...]

$ git branch work-newalgo
$ git checkout work-newalgo
Switched to branch 'work-newalgo'

$ git checkout -b work-betterdocs
Switched to a new branch 'work-betterdocs'
$ git branch
  master
* work-betterdocs
  work-newalgo
```

Git: Branches

```
[ make a bunch of commits on work branch ]

$ git checkout master
Switch to branch 'master'
$ git merge work-betterdocs
Updating 213a816..a9fae1e
Fast-forward
 README    |     100 +++++++++
 LICENSE   |       5 +
 INSTALL   | 120/30 ++++++++++++----
 3 files changed, 225 insertions(+), 30
  deletions(-)
 create mode 100644 LICENSE
 create mode 100644 README

$ git branch -d work-betterdocs
Deleted branch work-betterdocs (was a9fae1e).
```

Git: Collaboration

```
$ git clone git://github.com/jquery/jquery.git
[ ... ]
$ cd jquery
$ git remote
origin
$ git branch -a
master
remotes/origin/HEAD -> origin/master
remotes/origin/master
remotes/origin/mobile
remotes/origin/omgrequire
$ git fetch
[ ... ]
From git://github.com/jquery/jquery.git
   0a1b2c3..f9e8d7c  master -> origin/master
$ git merge origin/master
```

Git: Collaboration

```
$ git pull
```

Combine fetch and merge operations

```
$ git pull --rebase
```

“Rebase” your changes to pulled-in changes, preserving linearity of history (cf. `git rebase`).

```
$ git push
```

Update the remote's version of the branch with your local version.

```
$ git remote [...]
```

Manage the remotes known to your repository.

Git: Collaboration

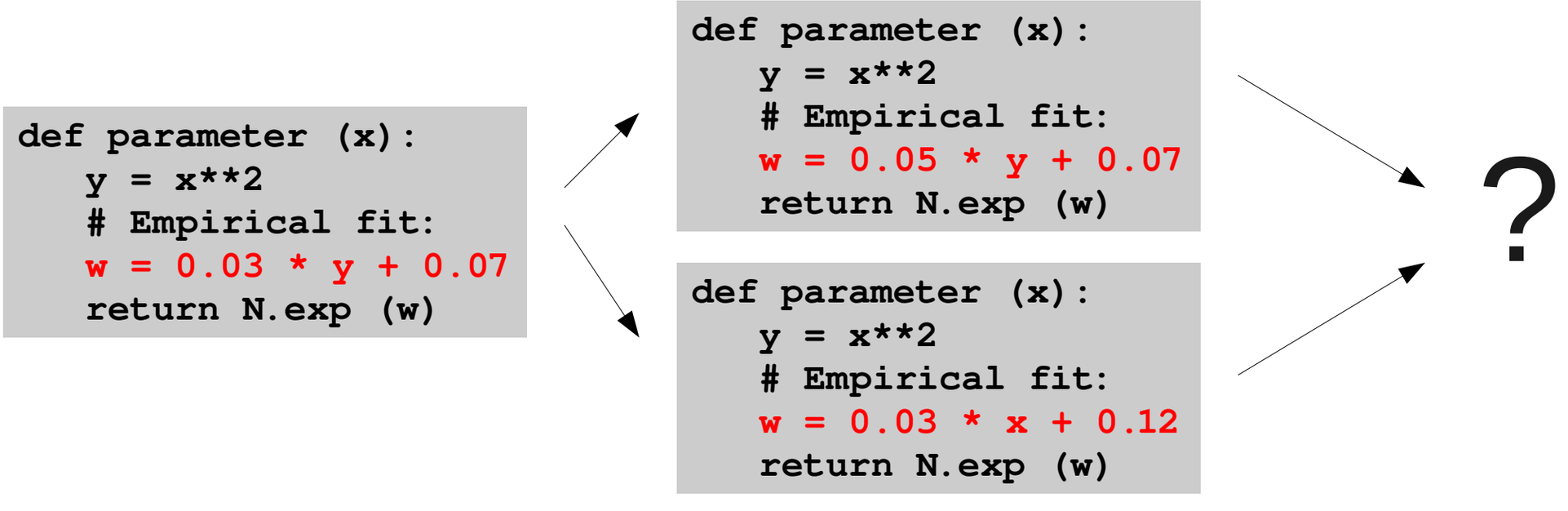
```
$ git clone /path/on/shared/disk

$ git clone git://git-server.com/...

$ git clone user@host:path/to/repo

$ git clone http://host/repo.git
```

```
$ cd ~/public_html
$ mkdir mycode.git
$ cd mycode.git
$ git init --bare
$ mv hooks/post-update.sample hooks/post-update # magic!

$ git clone http://dept.berkeley.edu/~user/mycode.git # public, readonly
$ git clone user@dept.berkeley.edu/~/public_html/mycode.git # priv, r-w
```

Fancy web frontends: `github.com`, `gitorious.org`

Git: Conflicts and Resolution

- Sometimes you want to merge two branches that modify the same part of the same file.
- Need to understand file's semantics to merge successfully: human intervention required



Git: Conflicts and Resolution

- Not enough time to explain. See:

http://book.git-scm.com/3_basic_branching_and_merging.html

http://weblog.masukomi.org/2008/07/12/handling-and-avoiding-conflicts-in-git

Other Version Control Systems

- Core usage is always the same; easy to pick up the basics of any similar system
- Subversion (“SVN”)
- CVS
- Mercurial (“hg”), Bazaar (“bzr”)
- ClearCase™, BitKeeper™, Perforce™

One Final Thought

```
$ git init
$ git add thesis.tex
$ git push backupdisk
```

Breakout Session

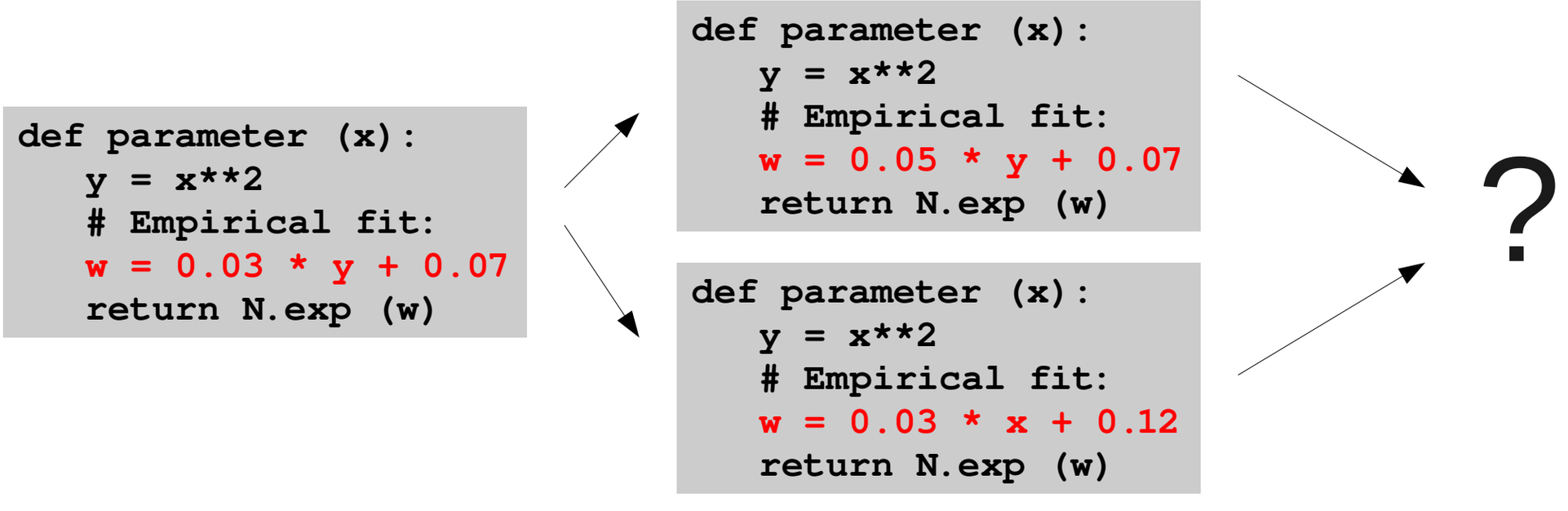
```
$ git clone git://github.com/pkgwdemo/bloomdemo.git
$ cd bloomdemo
$ less INSTRUCTIONS
```

Example fun with the index

```
$ emacs mystuff.py
$ git add -p mystuff.py
diff --git a/mystuff.py b/mystuff.py
index 99ad589..9f5e614 100644
--- a/mystuff.py
+++ b/mystuff.py
@@ -2,6 +2,9 @@

 print 'Hello, world!'

+import sys
+print sys.path
+
 for i in xrange (10):
     print i, '...'

Stage this hunk [y,n,q,a,d,/,j,J,g,e,?]? n
@@ -15,3 +18,5 @@ def parameter (x):

 print 'This is the end.'

+def square (y):
+    return [q**2 for q in y]
Stage this hunk [y,n,q,a,d,/,K,g,e,?]? y
```

Git: Conflicts and Resolution

- Sometimes you want to merge two branches that modify the same part of the same file.
- Need to understand file's semantics to merge successfully: human intervention required



Git: Conflicts and Resolution

```
$ git merge experiment
Auto-merging mystuff.py
CONFLICT (content): Merge conflict in mystuff.py
Automatic merge failed; fix conflicts and then commit the result.
$ git status
# On branch master
# Unmerged paths:
#   (use "git add/rm <file>..." as appropriate to mark resolution)
#
#       both modified:      mystuff.py
#
no changes added to commit (use "git add" and/or "git commit -a")
```

```
def parameter (x):
    y = x**2
    # Empirical fit:
<<<<<<< HEAD
    w = 0.05 * y + 0.07
=======
    w = 0.03 * x + 0.12
>>>>>>> experiment
    return N.exp (w)
```

```
$ git add mystuff.py
$ git commit
```

Why We Like Git

- Fundamental reason: extremely well-engineered, underlying theory is solid. (Turns out Linus knows what he's doing.)
- Rock-solid reliability
- Very, very fast
- Open-source and Free software
- Ergonomic
- Extremely powerful suite of tools
- Decentralized code-sharing model
- Active, committed developer community
- Secure

Git: Underlying model AKA sorry: math

- Essentially a random ID, yet well-defined for a given input
- Or: name of thing ⬌ value of thing
- 2**160 ~ 10**48 = really, really, really big
- Obviously there are also more user-friendly ways to refer to commits

Git: Underlying model AKA sorry: math

- Commits all identified by “SHA1 cryptographic hashes” = 160-bit numbers

- E.g., `0d30e664c0839392a0ec8c7c266e9e194b8bb7f6`

Cryptographic hash function F:
F: (any sequence of bytes) -> (number between 0 and 2**(big))
Such that, if
F (x) = y,
then
F (only a tiny bit different than x) = completely different than y

Git: Some of the Rest of the Iceberg

Help

```
$ git help [subcommand]
```

Massaging recent history

```
$ git commit --amend
$ git reset
$ git rebase [-i]
```

Spelunking older history

```
$ git log [-S]
$ git show [commit]
$ git blame [file]
```

Tagging special commits

```
$ git tag v1.2
$ git push --tags
```

Shell script support

```
$ git ls-files
$ git describe
$ git grep
$ git rev-list
```

Distributing

```
$ git archive |gzip >snapshot.tgz
```

Advanced Collaboration

```
$ git cherry-pick
$ git format-patch
$ git rerere
$ git mergetool
$ git send-email
$ git stash
```

Dealing With Inferior Tools

```
$ git cvsimport
$ git svn
```

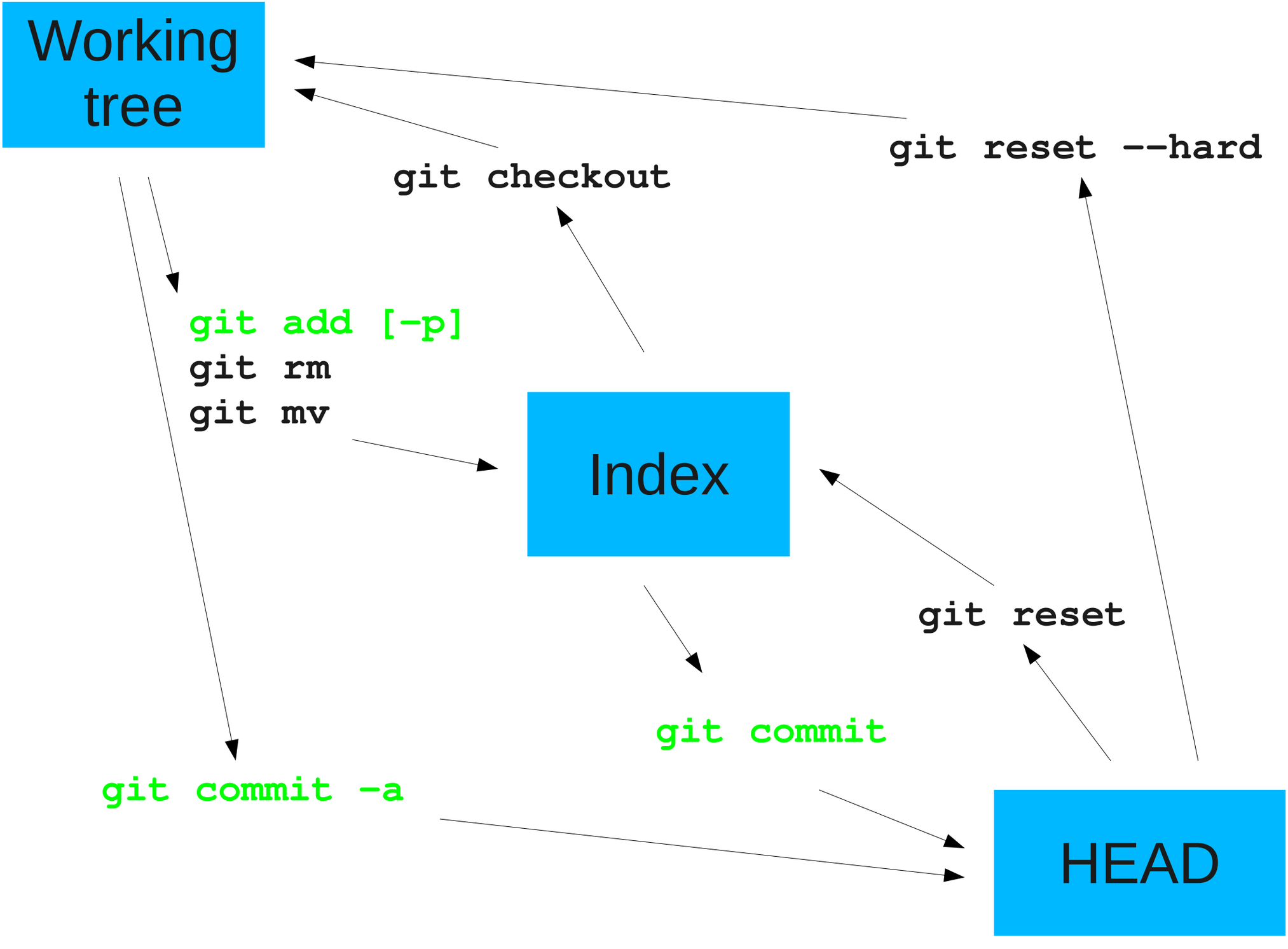
Nesting Repositories

```
$ git submodule
```

Diagnosing Bugs

```
$ git bisect
```

```
$ ls -1 $(git --exec-path) |wc -l
142
```


Working tree
git checkout
git reset --hard
git add [-p]
git rm
git mv
Index
git reset
git commit
git commit -a
HEAD

Git

- http://git-scm.com/
- Started by Linus Torvalds, original author of the Linux kernel
- Used by many major software projects
- A modern, “distributed” VCS