



Building useful project utils in Python

From spaghetti shell scripts to presentable Python

Johan Herland
@jherland
<jherland@cisco.com>
<johan@herland.net>

Prerequisites

- Basic familiarity with Unix shell scripting (`bash`)
- Some familiarity with `python3` (`python2` also works, but YMMV)

Outline

- Motivation
 - What scripts are we talking about?
 - Shell scripts! What are they good for?
 - Why is Python better?
- Shell → Python — Tips + Techniques
 - Code organization
 - Command-line interfaces in Python
 - Running other programs from Python
 - Manipulating the filesystem from Python
 - Writing unit tests in Python
 - A controversial take on error handling
- Case study — `binst`
 - A real-world rewrite from Shell to Python
 - Adding features that would have been impossible in Shell
- Summary
- Q & A

Motivation

What scripts are we talking about?

"The glue that holds your project together"

Developer tools

- Wrappers for more complex commands
- Workflow helpers

Packaging, deployment, release, ...

- Signing images
- Publishing assets

Build-time

- Small + simple build systems
- Build system helpers, plug-ins
- Code generators

In production

- Embedded targets
- Inside deployed containers

Shell scripts! What are they good for?

Pros

- Short and simple stuff
- Straightforward logic
- No data structures

Shell scripts! What are they good for?

Pros

- Short and simple stuff
- Straightforward logic
- No data structures

Cons

- Scripts grow, in length and complexity
- Complex logic
- Data structures
- Configurability
- Reusability
- Safety!

Why is Python better?

Showdown! Shell vs. Python

Why is Python better? — Syntax

Shell

- Reasonably easy to become productive
- Similar to command-line; skills are transferable
- Complexity → write-only
- Trivial (and necessary) to run other programs
- Many pitfalls, hard to become **safe**
- Many online resources/examples
- **Common practice != Best practice!**

```
filename="My favourite quotations.txt"
touch $filename
```

Why is Python better? — Syntax

Shell

- Reasonably easy to become productive
- Similar to command-line; skills are transferable
- Complexity → write-only
- Trivial (and necessary) to run other programs
- Many pitfalls, hard to become **safe**
- Many online resources/examples
- **Common practice != Best practice!**

```
filename="My favourite quotations.txt"
touch $filename
```

Python

- Simple and beginner-friendly syntax
- Considered one of the more readable languages
- Not as trivial to run other programs
- Fewer pitfalls than shell.
- Lots of literature and resources.
- Harder to come across "dangerous" examples

Why is Python better? — Data structures and types

Shell

- Everything is a string
- Whitespace in values often cause problems, and must be handled carefully
- Some support for integer arithmetic
- Some support for arrays

«Unlike many other programming languages, Bash does not segregate its variables by "type." Essentially, Bash variables are character strings, but, depending on context, Bash permits arithmetic operations and comparisons on variables. The determining factor is whether the value of a variable contains only digits.»

— Advanced Bash-Scripting Guide

Why is Python better? — Data structures and types

Shell

- Everything is a string
- Whitespace in values often cause problems, and must be handled carefully
- Some support for integer arithmetic
- Some support for arrays

«Unlike many other programming languages, Bash does not segregate its variables by "type." Essentially, Bash variables are character strings, but, depending on context, Bash permits arithmetic operations and comparisons on variables. The determining factor is whether the value of a variable contains only digits.»

— Advanced Bash-Scripting Guide

Python

- Dynamically (but strongly) typed
- Built-in types:
 - Numerics — `int`, `float`, `complex`
 - Sequences — `list`, `tuple`, `range`
 - Text and binary data — `str`, `bytes`, ...
 - Sets — `set`, ...
 - Mappings — `dict`
 - Others:
 - Modules
 - Classes and Class instances
 - Functions and Methods
 - Iterators and Generators
 - Exceptions
 - ...

Why is Python better? — Modularity

Shell

Within files: functions

- (sometimes non-obvious behavior, though, e.g. variables have global scope by default)

Between files: access in two ways:

1. Direct call (*think*: run as subprocess)
2. Use `source`
(*think*: `#include` — no namespaces!)

However, a script cannot easily figure out if it has been *called* or *sourced*, so scripts are typically **either** written for immediate usage **or** as a *bag of functions* reusable by sourcing (but does nothing when called).

Why is Python better? — Modularity

Shell

Within files: functions

- (sometimes non-obvious behavior, though, e.g. variables have global scope by default)

Between files: access in two ways:

1. Direct call (*think*: run as subprocess)
2. Use `source`
(*think*: `#include` — no namespaces!)

However, a script cannot easily figure out if it has been *called* or *sourced*, so scripts are typically **either** written for immediate usage **or** as *bag of functions* reusable by sourcing (but does nothing when called).

Python

Within files: functions, classes, data, ...

Between files: access in two ways:

1. `import`
2. Call via `subprocess`

Modules with `__main__` guards allows **both** for importability, and for immediate usage!

Why is Python better? — Modularity (cont.)

foo.sh:

```
#!/bin/sh

echo "This is always run"

foo() {
    echo "This is foo"
}
```

bar.sh:

```
#!/bin/sh

source foo.sh

foo()
```

Why is Python better? — Modularity (cont.)

foo.sh:

```
#!/bin/sh

echo "This is always run"

foo() {
    echo "This is foo"
}
```

bar.sh:

```
#!/bin/sh

source foo.sh

foo()
```

foo.py:

```
#!/usr/bin/env python3

print("This is always run")

def foo():
    print('This is foo')

if __name__ == '__main__':
    print("Skipped when this file is imported")
```

bar.py:

```
#!/usr/bin/env python3

import foo

foo.foo()
```

Why is Python better? — Tools & Libraries

Shell

- Language itself is very minimal
- Use system binaries from Unix ecosystem to do heavy lifting

Why is Python better? — Tools & Libraries

Shell

- Language itself is very minimal
- Use system binaries from Unix ecosystem to do heavy lifting

Python

- Very rich stdlib — can do most things "within" the language
- Lots more 3rd-party modules available (e.g. Python Package Index)
- Use `subprocess` module to call out to other binaries

Shell → Python

Tips + Techniques



Code organization

- Start by simply transcribing your shell script into Python!
 - One statement after another
 - No change in structure
 - Everything is run from top to bottom

Code organization

- Start by simply transcribing your shell script into Python!
 - One statement after another
 - No change in structure
 - Everything is run from top to bottom
- Refactor code into functions, classes, data structures, ...
 - Incrementally improve the structure of your script

Code organization

- Start by simply transcribing your shell script into Python!
 - One statement after another
 - No change in structure
 - Everything is run from top to bottom
- Refactor code into functions, classes, data structures, ...
 - Incrementally improve the structure of your script
- Add the `__main__` guard
 - Makes the functions/classes/data reusable from other scripts

Code organization

- Start by simply transcribing your shell script into Python!
 - One statement after another
 - No change in structure
 - Everything is run from top to bottom
- Refactor code into functions, classes, data structures, ...
 - Incrementally improve the structure of your script
- Add the `__main__` guard
 - Makes the functions/classes/data reusable from other scripts
- Add a `main()` function
 - Common convention in python
 - Other scripts can now `import + main()` instead of `subprocess.run()`

Code organization

- Start by simply transcribing your shell script into Python!
 - One statement after another
 - No change in structure
 - Everything is run from top to bottom
- Refactor code into functions, classes, data structures, ...
 - Incrementally improve the structure of your script
- Add the `__main__` guard
 - Makes the functions/classes/data reusable from other scripts
- Add a `main()` function
 - Common convention in python
 - Other scripts can now `import + main()` instead of `subprocess.run()`
- Split reusable functionality into separate/stand-alone modules
 - Add unit tests here to verify API + functionality

Command-line interfaces in Python

Follow Unix conventions:

- Exit code 0 → success, non-zero → failure
 - Default exit code from python is 0
 - Uncaught exceptions result in exit code 1
- Use `sys.stdin / sys.stdout` for default I/O
 - `print()` defaults to `sys.stdout`
- Log errors to `sys.stderr`
 - The `logging` module defaults to `sys.stderr`
- Mandatory command-line args are *positional*
- Optional command-line args use --dashes

Command-line interfaces in Python

Follow Unix conventions:

- Exit code 0 → success, non-zero → failure
 - Default exit code from python is 0
 - Uncaught exceptions result in exit code 1
- Use `sys.stdin / sys.stdout` for default I/O
 - `print()` defaults to `sys.stdout`
- Log errors to `sys.stderr`
 - The `logging` module defaults to `sys.stderr`
- Mandatory command-line args are *positional*
- Optional command-line args use --dashes

Use `argparse` ↓

Using argparse:

```
#!/usr/bin/env python3
'''Calculate the square of a given number.'''\n\nimport argparse\nimport sys\n\n\ndef calc(num, exp):\n    return num ** exp\n\n\ndef main():\n    parser = argparse.ArgumentParser(description=sys.modules[__name__].__doc__)\n    parser.add_argument(\n        'num', type=int,\n        help='the number to be squared')\n    parser.add_argument(\n        '--cube', '-3', action='store_true',\n        help='calculate the cube (instead of square)')\n    parser.add_argument(\n        '--file', '-f', type=argparse.FileType('w'), default=sys.stdout,\n        help='write result here (stdout by default)')\n\n    args = parser.parse_args()\n    result = calc(args.num, 3 if args.cube else 2)\n    print(result, file=args.file)\n\nif __name__ == '__main__':\n    main()
```

Using argparse:

```
#!/usr/bin/env python3
'''Calculate the square of a given number.'''
import argparse
import sys

def calc(num, exp):
    return num ** exp

def main():
    parser = argparse.ArgumentParser(description=sys.modules[__name__].__doc__)
    parser.add_argument(
        'num', type=int,
        help='the number to be squared')
    parser.add_argument(
        '--cube', '-3', action='store_true',
        help='calculate the cube (instead of square)')
    parser.add_argument(
        '--file', '-f', type=argparse.FileType('w'), default=sys.stdout,
        help='write result here (stdout by default)')

    args = parser.parse_args()
    result = calc(args.num, 3 if args.cube else 2)
    print(result, file=args.file)

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

```
# <- Module docstring
#     (.__doc__ on the module)

# <- Reusable function:
import myscript
myscript.calc(123, 2)

# <- Reuse module docstring
#     in usage help text
# <- Positional integer arg

# <- Optional arg, boolean,
#     default: False
# <- Optional arg, open file

# <- __main__ guard
```

```
$ ./myscript.py
usage: myscript.py [-h] [--cube] [--file FILE] num
myscript.py: error: the following arguments are required: num
```

```
$ ./myscript.py  
usage: myscript.py [-h] [--cube] [--file FILE] num  
myscript.py: error: the following arguments are required: num
```

```
$ ./myscript.py -h  
usage: myscript.py [-h] [--cube] [--file FILE] num
```

Calculate the square of a given number.

positional arguments:

num	the number to be squared
-----	--------------------------

optional arguments:

-h, --help	show this help message and exit
--cube, -3	calculate the cube (instead of square)
--file FILE, -f FILE	write result here (stdout by default)

```
$ ./myscript.py  
usage: myscript.py [-h] [--cube] [--file FILE] num  
myscript.py: error: the following arguments are required: num
```

```
$ ./myscript.py -h  
usage: myscript.py [-h] [--cube] [--file FILE] num
```

Calculate the square of a given number.

positional arguments:

num the number to be squared

optional arguments:

-h, --help show this help message and exit

--cube, -3 calculate the cube (instead of square)

--file FILE, -f FILE write result here (stdout by default)

```
$ ./myscript.py 5
```

25

```
$ ./myscript.py  
usage: myscript.py [-h] [--cube] [--file FILE] num  
myscript.py: error: the following arguments are required: num
```

```
$ ./myscript.py -h  
usage: myscript.py [-h] [--cube] [--file FILE] num
```

Calculate the square of a given number.

positional arguments:

num the number to be squared

optional arguments:

-h, --help show this help message and exit

--cube, -3 calculate the cube (instead of square)

--file FILE, -f FILE write result here (stdout by default)

```
$ ./myscript.py 5  
25
```

```
$ ./myscript.py 5 -3  
125
```

```
$ ./myscript.py  
usage: myscript.py [-h] [--cube] [--file FILE] num  
myscript.py: error: the following arguments are required: num
```

```
$ ./myscript.py -h  
usage: myscript.py [-h] [--cube] [--file FILE] num
```

Calculate the square of a given number.

positional arguments:

 num the number to be squared

optional arguments:

 -h, --help show this help message and exit

 --cube, -3 calculate the cube (instead of square)

 --file FILE, -f FILE write result here (stdout by default)

```
$ ./myscript.py 5
```

```
25
```

```
$ ./myscript.py 5 -3
```

```
125
```

```
$ ./myscript.py 10 -3 -f foo  
$ cat foo  
1000
```

Running other programs from Python — subprocess

Simple cases

```
import subprocess

# Prepare command as a list of strings
argv = ['ping', '-c1', 'google.com']

# Run command, get its exit code
exitcode = subprocess.run(argv).returncode

# Run command, raise exception on non-zero exit code
subprocess.run(argv, check=True)

# Run command, raise exception if it fails, capture its output
result = subprocess.run(argv, check=True, capture_output=True)
output, errors = result.stdout, result.stderr
```

Running other programs from Python — subprocess

Simple cases

```
import subprocess

# Prepare command as a list of strings
argv = ['ping', '-c1', 'google.com']

# Run command, get its exit code
exitcode = subprocess.run(argv).returncode

# Run command, raise exception on non-zero exit code
subprocess.run(argv, check=True)

# Run command, raise exception if it fails, capture its output
result = subprocess.run(argv, check=True, capture_output=True)
output, errors = result.stdout, result.stderr
```

More complex cases

See <https://docs.python.org/3/library/subprocess.html>

Manipulating the filesystem from Python — `pathlib`

- Features:
 - Creating/moving/removing files and dirs
 - Looking for specific (types of) files
 - Handling *relative* vs *absolute* paths
- Replaces/supersedes:
 - Parts of `os.path.*` and elsewhere in `os.*`
 - `fnmatch.fnmatch()` and `glob.glob()`
- <https://docs.python.org/3/library/pathlib.html>

Other modules

- `os` and `os.path`
- `shutil`

Manipulating the filesystem from Python — pathlib

- Features:
 - Creating/moving/removing files and dirs
 - Looking for specific (types of) files
 - Handling *relative* vs *absolute* paths
- Replaces/supersedes:
 - Parts of `os.path.*` and elsewhere in `os.*`
 - `fnmatch.fnmatch()` and `glob.glob()`
- <https://docs.python.org/3/library/pathlib.html>

```
from pathlib import Path

mydir = Path.cwd() / 'mydir'
mydir.mkdir(exist_ok=True)

foo = mydir / 'foo.txt'
if foo.exists():
    foo.rename(foo.with_suffix('.backup'))

with foo.open('w') as f:
    for path in mydir.iterdir():
        print(path.relative_to(mydir), file=f)

for path in mydir.glob('*.*.backup'):
    path.unlink()
```

Other modules

- `os` and `os.path`
- `shutil`

Writing unit tests in Python

What to test

- As much as possible
- Certainly reusable functionality

Available frameworks

- `doctest`
- `unittest`
- `pytest` (not in the standard library)
- Many others...

Writing unit tests in Python

What to test

- As much as possible
- Certainly reusable functionality

Available frameworks

- `doctest`
- `unittest`
- `pytest` (not in the standard library)
- Many others...

Note

This is not a talk on test frameworks or unit testing in general.

For everything but the smallest scripts, put your tests in separate files, or even separate directories, but here we'll look at how you can put small tests in the same file as the rest of the script.

Remember?

```
#!/usr/bin/env python3
'''Calculate the square of a given number.''''

import argparse
import sys

def calc(num, exp):
    return num ** exp

def main():
    parser = argparse.ArgumentParser(description=sys.modules[__name__].__doc__)
    parser.add_argument(
        'num', type=int,
        help='the number to be squared')
    parser.add_argument(
        '--cube', '-3', action='store_true',
        help='calculate the cube (instead of square)')
    parser.add_argument(
        '--file', '-f', type=argparse.FileType('w'), default=sys.stdout,
        help='write result here (stdout by default)')

    args = parser.parse_args()
    result = calc(args.num, 3 if args.cube else 2)
    print(result, file=args.file)

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

Writing unit tests in Python — doctest

Ad-hoc testing:

```
$ python
Python 3.7.0 (default, Jul 15 2018, 10:44:58)
[...]
>>> import myscript
>>> myscript.calc(3, 2)
9
```

Let's formalize that a bit:

```
def calc(num, exp):
    '''Return 'num' raised to the power of 'exp'.

    >>> [calc(0, 2), calc(1, 2), calc(2, 2)]
    [0, 1, 4]
    >>> [calc(0, 3), calc(1, 3), calc(2, 3)]
    [0, 1, 8]
    '''

    return num ** exp
```

Writing unit tests in Python — doctest

Ad-hoc testing:

```
$ python
Python 3.7.0 (default, Jul 15 2018, 10:44:58)
[...]
>>> import myscript
>>> myscript.calc(3, 2)
9
```

Let's formalize that a bit:

```
def calc(num, exp):
    '''Return 'num' raised to the power of 'exp'.

    >>> [calc(0, 2), calc(1, 2), calc(2, 2)]
    [0, 1, 4]
    >>> [calc(0, 3), calc(1, 3), calc(2, 3)]
    [0, 1, 8]
    '''

    return num ** exp
```

Run tests from command-line:

```
$ python -m doctest myscript.py -v
Trying:
    [calc(0, 2), calc(1, 2), calc(2, 2)]
Expecting:
    [0, 1, 4]
ok
Trying:
    [calc(0, 3), calc(1, 3), calc(2, 3)]
Expecting:
    [0, 1, 8]
ok
2 items had no tests:
    myscript
    myscript.main
1 items passed all tests:
    2 tests in myscript.calc
2 tests in 3 items.
2 passed and 0 failed.
Test passed.
```

Run tests from within the script:

```
if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

Writing unit tests in Python — unittest

```
import unittest

...

class TestSquare(unittest.TestCase):

    def test_zero_squared_is_zero(self):
        self.assertEqual(0, calc(0, 2))

    def test_one_squared_is_one(self):
        self.assertEqual(1, calc(1, 2))

    def test_two_squared_is_four(self):
        self.assertEqual(4, calc(2, 2))

class TestCube(unittest.TestCase):

    def test_zero_cubed_is_zero(self):
        self.assertEqual(0, calc(0, 3))

    def test_one_cubed_is_one(self):
        self.assertEqual(1, calc(1, 3))

    def test_two_cubed_is_eight(self):
        self.assertEqual(8, calc(2, 3))
```

Writing unit tests in Python — unittest

```
import unittest

...
class TestSquare(unittest.TestCase):

    def test_zero_squared_is_zero(self):
        self.assertEqual(0, calc(0, 2))

    def test_one_squared_is_one(self):
        self.assertEqual(1, calc(1, 2))

    def test_two_squared_is_four(self):
        self.assertEqual(4, calc(2, 2))

class TestCube(unittest.TestCase):

    def test_zero_cubed_is_zero(self):
        self.assertEqual(0, calc(0, 3))

    def test_one_cubed_is_one(self):
        self.assertEqual(1, calc(1, 3))

    def test_two_cubed_is_eight(self):
        self.assertEqual(8, calc(2, 3))
```

Run tests from command-line:

```
$ python -m unittest myscript.py
.....
-----
Ran 6 tests in 0.000s
OK
```

Run tests from within the script:

```
if __name__ == '__main__':
    unittest.main()
```

Writing unit tests in Python — pytest

```
def test_calc():
    vectors = [
        (0, 2, 0), (1, 2, 1), (2, 2, 4),
        (0, 3, 0), (1, 3, 1), (2, 3, 8),
    ]
    for num, exp, expect in vectors:
        assert calc(num, exp) == expect
```

Writing unit tests in Python — pytest

```
def test_calc():
    vectors = [
        (0, 2, 0), (1, 2, 1), (2, 2, 4),
        (0, 3, 0), (1, 3, 1), (2, 3, 8),
    ]
    for num, exp, expect in vectors:
        assert calc(num, exp) == expect
```

Run tests from command-line:

```
$ pytest myscript.py
===== test session starts =====
[...]
collected 1 item

myscript.py . [100%]

===== 1 passed in 0.01 seconds =====
```

Run tests from within the script:

```
if __name__ == '__main__':
    import pytest
    pytest.main([__file__])
```

A controversial take on error handling

A controversial take on error handling

Don't!

A controversial take on error handling

Don't!

- Sloppy error handling is worse than *no* error handling
- Python tracebacks are actually useful for debugging
- Simplify and remove error handling that might otherwise obscure the source of the error
- Allow Python exceptions to *escape* and terminate the program
- Guarantees non-zero exit code

Catching exceptions, and replacing them with a (generic) error message often hides useful details about the error, whereas the traceback points directly at a relevant line of code, and often includes the value that triggered the error.

A controversial take on error handling

Don't!

- Sloppy error handling is worse than *no* error handling
- Python tracebacks are actually useful for debugging
- Simplify and remove error handling that might otherwise obscure the source of the error
- Allow Python exceptions to *escape* and terminate the program
- Guarantees non-zero exit code

Catching exceptions, and replacing them with a (generic) error message often hides useful details about the error, whereas the traceback points directly at a relevant line of code, and often includes the value that triggered the error.

That said, ...

Simple usage/user errors are best handled with a simple error message (but `argparse` gives you this for free).

Tracebacks may not work as well if the distance between script users and script maintainers is large.

Case study

binst

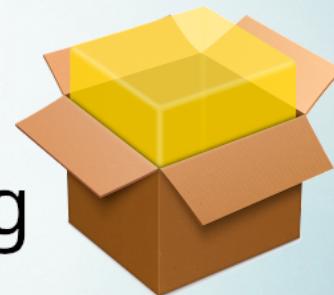
Common developer workflow



build



.pkg



binst



binst shell script

```

1 #!/bin/bash
2 # Shows a little about this and that to speed up test software
3 # uploads
4 # uploads
5 # uploads
6 self+="basename $0"
7 toplevel=readlink -f "${dirname}"$0"/.."
8 help=
9 listtargets=
10 remotetest=
11 remotetest_install=
12 verbose=
13 cat_cmdbot=
14 nc_cmdbot=
15 scp_cmdbotscp=
16 postbin_cmdbot=
17 sunos_cmdbot=
18 installimageargs=
19 fromtoplevel=
20 objdir=
21 show_prods() {
22     cat <>EOF
23     asterix
24     asterix_apps
25     asterix_gui
26     asterix_cmdbot
27     asterix_cmdbotsvc
28     asterix_slaveapps
29     parents=
30     parents_cmdbot=
31     carbon
32     carbon_cmdbot=
33     carbon_cmdbotsvc
34     cehost
35     dritsch_apps
36     dritsch_cmdbot=
37     dritsch_gui
38     dritsch_cmdbotsvc
39     halley
40     halley_cmdbotsvc
41     iderix
42     mody
43     pyramid
44     pyramid_cmdbot=
45     pyramid_cmdbotsvc
46     sunrise_cmdbot=
47     tempo
48     zenith_cmdbot=
49     EOF
50 }
51 usobjl() {
52     cat <>EOF
53     Standard binst using netcat for efficiency:
54     $self [-f <file>] [-v -d] [<-e args>] <-product> <destination>
55
56     Via another host:
57     $self [-f <file>] [-v -d] <product> <destination>
58
59     $self [-f <file>] <host> [-v -d] [<-e args>] <-product> <destination>
60
61     List all available target types:
62     $self --list
63
64     Full option list:
65     -f|--objdir - Pick install file from this path
66     -d|--dest - Installs a specific image from the file specified.
67     -v|--verbose - Shows verbose file transfer. Requires pv(1).
68     -y|--allow-test-software - Allow installing test sw on releases installs.
69     -z|--zenith - Set zenith as the default target. (IPv4/IPv6/IPv6LL)
70     -e|--args - Add these extra arguments to installimage on target.
71     -t|--products - Install image for this product type. May be repeated.
72     -r|--remotetest - Run remotetest on the target.
73     -l|--list-targets - Show supported targets for binst.
74     -h|--help - Print this message and exit.
75     EOF
76 }
77
78 ls_ipv6() {
79     [[ "$1" == ":" ]] && read 0
80     [[ "$1" == ":" ]] && echo -e '\000'
81     [[ "$1" == ":" ]] && echo -e '\000'
82     [[ "$1" == ":" ]] && echo -e '\000'
83     [[ "$1" == ":" ]] && echo -e '\000'
84 }
85
86 ls_linklocal() {
87     [[ "$1" == ":" ]] && read 0
88     [[ "$1" == ":" ]] && echo -e '\000'
89     [[ "$1" == ":" ]] && echo -e '\000'
90     [[ "$1" == ":" ]] && echo -e '\000'
91     [[ "$1" == ":" ]] && echo -e '\000'
92     [[ "$1" == ":" ]] && echo -e '\000'
93     [[ "$1" == ":" ]] && echo -e '\000'
94     while [ -n "$1" ]; do
95         case "$1" in
96             -l|--list-targets)
97                 listtargets=
98             ;;
99             -h|--help)
100                 help=
101             ;;
102             -f|--file)
103                 file=
104                 if [ -z "$1" ]; then
105                     fail=1
106                 fi
107                 if [ "$file" ]; then
108                     userfile="$1"
109                 fi
110             ;;
111             -o|--objdir)
112                 objdir="$1"
113                 if [ -z "$1" ]; then
114                     fail=1
115                 fi
116             ;;
117             -t|--target)
118                 shift
119                 if [ -z "$1" ]; then
120                     fail=1
121                 fi
122                 target=$1
123                 if [ "$target" == "asterix" ]; then
124                     target+="gui"
125                 fi
126                 ->installimageargs()
127                 shift
128                 if [ -z "$1" ]; then
129                     fail=1
130                 fi
131                 help=
132                 if [ "$target" == "asterix" ]; then
133                     installimageargs="$1"
134                     ->allow-test-software()
135                     installimageargs+=<file>
136                     if [ "$target" == "asterix" ]; then
137                         ->unprodroid()
138                         shift
139                         if [ -z "$1" ]; then
140                             verbose=
141                         verbose+=1
142                         if [ "$target" == "asterix" ]; then
143                             if [ "$1" == "command -v pv" ]; then
144                                 cat_cmdbot
145                             else
146                                 echo "pv(1) is not installed. Cannot be verbose during file transfer."
147                                 fail=1
148                             fi
149                         fi
150                         ->4()
151                         nc_cmdbot=
152                         nc_cmdbotsvc=
153                         ->vbox()
154                         shift
155                         if [ -z "$1" ]; then
156                             fail=1
157                         fi
158                         if [ "$target" == "asterix" ]; then
159                             ->tilera()
160                             shift
161                             if [ -z "$1" ]; then
162                                 if [ ! -n "$dest" ]; then
163                                     echo "Cannot specify multiple destinations ($dest and $1)"
164                                 fi
165                                 help=
166                                 dest="$1"
167                             fi
168                         else
169                             ->xe()
170                             shift
171                             if [ -z "$1" ]; then
172                                 if [ ! -n "$dest" ]; then
173                                     echo "Via another host"
174                                     done
175                                     shift
176                                     if [ -n "$listtargets" ]; then
177                                         shift
178                                         if [ -z "$1" ]; then
179                                             if [ ! -n "$targetarray" ] || eq "0"; then
180                                                 fail=1
181                                             fi
182                                             if [ "$target" == "asterix" ]; then
183                                                 ->astertx()
184                                                 targetarray+=<file> -a ${#targetarray[@]} -ne "1" ; then
185                                                 fail=1
186                                             fi
187                                             ->prodroid()
188                                             ->carbon()
189                                             if [ -n "$help" ]; then
190                                                 if [ -n "$target" ]; then
191                                                     if [ -n "$dest" ]; then
192                                                         exit 1
193                                                     else
194                                                         exit 0
195                                                     fi
196                                                 fi
197                                                 ssh_desktop="ssh -S $dest"
198                                                 ssh_desktop+=<file>
199                                                 ssh_desktop+=<file>
200                                                 if [ ! -n "$dest" ]; then
201                                                     ssh_desktop+=<file> -a <file> ; then
202                                                     ssh_desktop+=<file> -S $dest
203                                                     if [ "$target" == "asterix" ]; then
204                                                         ssh_desktop+="<file>_dest"
205                                                     fi
206                                                 fi
207                                                 ssh_cmdbot="ssh -S $dest"
208                                                 ssh_cmdbot+=<file>
209                                                 ssh_cmdbot+=<file>
210                                                 declare -a final_args
211                                                 for target in ${targetarray[@]}; do
212                                                     ssh root@$target -S <file> -o StrictHostKeyChecking=no -o "UserKnownHostsFile=/dev/null" ${final_args[@]}
213                                                 done
214                                                 ssh_cmdbot+="<file>_dest"
215                                                 if [ "$target" == "asterix" ]; then
216                                                     declare -a final_args
217                                                     for target in ${targetarray[@]}; do
218                                                         if [ "$target" == "asterix" ]; then
219                                                             filters=
220                                                             filters+=<file>
221                                                             filters+=<file>
222                                                             filters+=<file>
223                                                             filters+=<file>
224                                                             echo "Installing $target"
225                                                             case "$target" in
226                                                               asterix)
227                                                               desc="Asterix Complete Image, target arm linux"
228                                                               ;;
229                                                               asterix_gui)
230                                                               desc="Asterix GUI code, target arm linux"
231                                                               targetpath=/mnt/base/applications/code, target arm linux
232                                                               targetpath=/mnt/base/active/apps.img
233                                                               postbincmd="/bin/main/restart update"
234                                                               asterix_gui
235                                                               buildtarget="asterix.gui"
236                                                               desc="Asterix GUI code, target arm linux"
237                                                               targetpath=/mnt/base/active/gui.img
238                                                               targetpath=/finaltargetpath.tmp
239                                                               postbinbind="--exec /int/d$1$ &> /dev/null 2>&1 &&
240                                                               targetpath=/finaltargetpath &&
241                                                               targetpath=/finaltargetpath &&
242                                                               targetpath=/finaltargetpath &&
243                                                               targetpath=/finaltargetpath &&
244                                                               targetpath=/finaltargetpath &&
245                                                               targetpath=/finaltargetpath &&
246                                                               targetpath=/finaltargetpath &&
247                                                               targetpath=/finaltargetpath &&
248                                                               targetpath=/finaltargetpath &&
249                                                               targetpath=/finaltargetpath &&
250                                                               targetpath=/finaltargetpath &&
251                                                               targetpath=/finaltargetpath &&
252                                                               targetpath=/finaltargetpath &&
253                                                               targetpath=/finaltargetpath &&
254                                                               targetpath=/finaltargetpath &&
255                                                               targetpath=/finaltargetpath &&
256                                                               targetpath=/finaltargetpath &&
257                                                               targetpath=/finaltargetpath &&
258                                                               targetpath=/finaltargetpath &&
259                                                               targetpath=/finaltargetpath &&
260                                                               targetpath=/finaltargetpath &&
261                                                               targetpath=/finaltargetpath &&
262                                                               targetpath=/finaltargetpath &&
263                                                               targetpath=/finaltargetpath &&
264                                                               targetpath=/finaltargetpath &&
265                                                               targetpath=/finaltargetpath &&
266                                                               targetpath=/finaltargetpath &&
267                                                               targetpath=/finaltargetpath &&
268                                                               targetpath=/finaltargetpath &&
269                                                               targetpath=/finaltargetpath &&
270                                                               targetpath=/finaltargetpath &&
271                                                               targetpath=/finaltargetpath &&
272                                                               targetpath=/finaltargetpath &&
273                                                               targetpath=/finaltargetpath &&
274                                                               targetpath=/finaltargetpath &&
275                                                               targetpath=/finaltargetpath &&
276                                                               targetpath=/finaltargetpath &&
277                                                               targetpath=/finaltargetpath &&
278                                                               targetpath=/finaltargetpath &&
279                                                               targetpath=/finaltargetpath &&
280                                                               targetpath=/finaltargetpath &&
281                                                               targetpath=/finaltargetpath &&
282                                                               targetpath=/finaltargetpath &&
283                                                               targetpath=/finaltargetpath &&
284                                                               targetpath=/finaltargetpath &&
285                                                               targetpath=/finaltargetpath &&
286                                                               targetpath=/finaltargetpath &&
287                                                               targetpath=/finaltargetpath &&
288                                                               targetpath=/finaltargetpath &&
289                                                               targetpath=/finaltargetpath &&
290                                                               targetpath=/finaltargetpath &&
291                                                               targetpath=/finaltargetpath &&
292                                                               targetpath=/finaltargetpath &&
293                                                               targetpath=/finaltargetpath &&
294                                                               targetpath=/finaltargetpath &&
295                                                               targetpath=/finaltargetpath &&
296                                                               targetpath=/finaltargetpath &&
297                                                               targetpath=/finaltargetpath &&
298                                                               targetpath=/finaltargetpath &&
299                                                               targetpath=/finaltargetpath &&
300                                                               targetpath=/finaltargetpath &&
301                                                               targetpath=/finaltargetpath &&
302                                                               targetpath=/finaltargetpath &&
303                                                               targetpath=/finaltargetpath &&
304                                                               targetpath=/finaltargetpath &&
305                                                               targetpath=/finaltargetpath &&
306                                                               targetpath=/finaltargetpath &&
307                                                               targetpath=/finaltargetpath &&
308                                                               targetpath=/finaltargetpath &&
309                                                               targetpath=/finaltargetpath &&
310                                                               targetpath=/finaltargetpath &&
311                                                               targetpath=/finaltargetpath &&
312                                                               targetpath=/finaltargetpath &&
313                                                               targetpath=/finaltargetpath &&
314                                                               targetpath=/finaltargetpath &&
315                                                               targetpath=/finaltargetpath &&
316                                                               targetpath=/finaltargetpath &&
317                                                               targetpath=/finaltargetpath &&
318                                                               targetpath=/finaltargetpath &&
319                                                               targetpath=/finaltargetpath &&
320                                                               targetpath=/finaltargetpath &&
321                                                               targetpath=/finaltargetpath &&
322                                                               targetpath=/finaltargetpath &&
323                                                               targetpath=/finaltargetpath &&
324                                                               targetpath=/finaltargetpath &&
325                                                               targetpath=/finaltargetpath &&
326                                                               targetpath=/finaltargetpath &&
327                                                               targetpath=/finaltargetpath &&
328                                                               targetpath=/finaltargetpath &&
329                                                               targetpath=/finaltargetpath &&
330                                                               targetpath=/finaltargetpath &&
331                                                               targetpath=/finaltargetpath &&
332                                                               targetpath=/finaltargetpath &&
333                                                               targetpath=/finaltargetpath &&
334                                                               targetpath=/finaltargetpath &&
335                                                               targetpath=/finaltargetpath &&
336                                                               targetpath=/finaltargetpath &&
337                                                               targetpath=/finaltargetpath &&
338                                                               targetpath=/finaltargetpath &&
339                                                               targetpath=/finaltargetpath &&
340                                                               targetpath=/finaltargetpath &&
341                                                               targetpath=/finaltargetpath &&
342                                                               targetpath=/finaltargetpath &&
343                                                               targetpath=/finaltargetpath &&
344                                                               targetpath=/finaltargetpath &&
345                                                               targetpath=/finaltargetpath &&
346                                                               targetpath=/finaltargetpath &&
347                                                               targetpath=/finaltargetpath &&
348                                                               targetpath=/finaltargetpath &&
349                                                               targetpath=/finaltargetpath &&
350                                                               targetpath=/finaltargetpath &&
351                                                               targetpath=/finaltargetpath &&
352                                                               targetpath=/finaltargetpath &&
353                                                               targetpath=/finaltargetpath &&
354                                                               targetpath=/finaltargetpath &&
355                                                               targetpath=/finaltargetpath &&
356                                                               targetpath=/finaltargetpath &&
357                                                               targetpath=/finaltargetpath &&
358                                                               targetpath=/finaltargetpath &&
359                                                               targetpath=/finaltargetpath &&
360                                                               targetpath=/finaltargetpath &&
361                                                               targetpath=/finaltargetpath &&
362                                                               targetpath=/finaltargetpath &&
363                                                               targetpath=/finaltargetpath &&
364                                                               targetpath=/finaltargetpath &&
365                                                               targetpath=/finaltargetpath &&
366                                                               targetpath=/finaltargetpath &&
367                                                               targetpath=/finaltargetpath &&
368                                                               targetpath=/finaltargetpath &&
369                                                               targetpath=/finaltargetpath &&
370                                                               targetpath=/finaltargetpath &&
371                                                               targetpath=/finaltargetpath &&
372                                                               targetpath=/finaltargetpath &&
373                                                               targetpath=/finaltargetpath &&
374                                                               targetpath=/finaltargetpath &&
375                                                               targetpath=/finaltargetpath &&
376                                                               targetpath=/finaltargetpath &&
377                                                               targetpath=/finaltargetpath &&
378                                                               targetpath=/finaltargetpath &&
379                                                               targetpath=/finaltargetpath &&
380                                                               targetpath=/finaltargetpath &&
381                                                               targetpath=/finaltargetpath &&
382                                                               targetpath=/finaltargetpath &&
383                                                               targetpath=/finaltargetpath &&
384                                                               targetpath=/finaltargetpath &&
385                                                               targetpath=/finaltargetpath &&
386                                                               targetpath=/finaltargetpath &&
387                                                               targetpath=/finaltargetpath &&
388                                                               targetpath=/finaltargetpath &&
389                                                               targetpath=/finaltargetpath &&
390                                                               targetpath=/finaltargetpath &&
391                                                               targetpath=/finaltargetpath &&
392                                                               targetpath=/finaltargetpath &&
393                                                               targetpath=/finaltargetpath &&
394                                                               targetpath=/finaltargetpath &&
395                                                               targetpath=/finaltargetpath &&
396                                                               targetpath=/finaltargetpath &&
397                                                               targetpath=/finaltargetpath &&
398                                                               targetpath=/finaltargetpath &&
399                                                               targetpath=/finaltargetpath &&
400                                                               targetpath=/finaltargetpath &&
401                                                               targetpath=/finaltargetpath &&
402                                                               targetpath=/finaltargetpath &&
403                                                               targetpath=/finaltargetpath &&
404                                                               targetpath=/finaltargetpath &&
405                                                               targetpath=/finaltargetpath &&
406                                                               targetpath=/finaltargetpath &&
407                                                               targetpath=/finaltargetpath &&
408                                                               targetpath=/finaltargetpath &&
409                                                               targetpath=/finaltargetpath &&
410                                                               targetpath=/finaltargetpath &&
411                                                               targetpath=/finaltargetpath &&
412                                                               targetpath=/finaltargetpath &&
413                                                               targetpath=/finaltargetpath &&
414                                                               targetpath=/finaltargetpath &&
415                                                               targetpath=/finaltargetpath &&
416                                                               targetpath=/finaltargetpath &&
417                                                               targetpath=/finaltargetpath &&
418                                                               targetpath=/finaltargetpath &&
419                                                               targetpath=/finaltargetpath &&
420                                                               targetpath=/finaltargetpath &&
421                                                               targetpath=/finaltargetpath &&
422                                                               targetpath=/finaltargetpath &&
423                                                               targetpath=/finaltargetpath &&
424                                                               targetpath=/finaltargetpath &&
425                                                               targetpath=/finaltargetpath &&
426                                                               targetpath=/finaltargetpath &&
427                                                               targetpath=/finaltargetpath &&
428                                                               targetpath=/finaltargetpath &&
429                                                               targetpath=/finaltargetpath &&
430                                                               targetpath=/finaltargetpath &&
431                                                               targetpath=/finaltargetpath &&
432                                                               targetpath=/finaltargetpath &&
433                                                               targetpath=/finaltargetpath &&
434                                                               targetpath=/finaltargetpath &&
435                                                               targetpath=/finaltargetpath &&
436                                                               targetpath=/finaltargetpath &&
437                                                               targetpath=/finaltargetpath &&
438                                                               targetpath=/finaltargetpath &&
439                                                               targetpath=/finaltargetpath &&
440                                                               targetpath=/finaltargetpath &&
441                                                               targetpath=/finaltargetpath &&
442                                                               targetpath=/finaltargetpath &&
443                                                               targetpath=/finaltargetpath &&
444                                                               targetpath=/finaltargetpath &&
445                                                               targetpath=/finaltargetpath &&
446                                                               targetpath=/finaltargetpath &&
447                                                               targetpath=/finaltargetpath &&
448                                                               targetpath=/finaltargetpath &&
449                                                               targetpath=/finaltargetpath &&
450                                                               targetpath=/finaltargetpath &&
```

binst shell script

Stats

- 450 lines of shell (`bash`)
- Helper functions
- Command-line interface (~50%)
- Data structure (almost) (~30%)
- Some logic:
 - `while` loop for command-line argument parsing
 - `main` `for` loop for binsting multiple targets
 - Lots of `if` / `elif` / `else` for error handling and fallbacks.
- Ends up here in most cases:

```
 ${cat_cmd} ${file} | ssh_wrapper root@"${ssh_dest}" ". /etc/profile; \
 ${installimage} -k /mnt/base/active/rk -f - $installimageargs; \
 [ -n \"${installimage[1]}\"] && \
 ${installimage[1]} -k /mnt/base/active/rk -f - $installimageargs"
```

binst shell script

Problems

- Complex shell script
- binst upgrades are different from in-the-field upgrades:
 - binst *pushes* .pkg onto the system
 - binst bypasses most of the upgrade machinery on the system
 - In the field, the system is told where to find the .pkg and *pulls* it from there.
- New requirement: Support systems with multiple devices

System with multiple devices

Simple systems have the camera built-in to the *codec* (device running the CE S/W)



System with multiple devices

Simple systems have the camera built-in to the *codec* (device running the CE S/W)



Bigger systems have codec running separately from one or more cameras



System with multiple devices

- System comprises codec plus one or more peripherals
 - Upgrading one, but not the other, likely breaks the connection
- Codec plus peripherals must run same/compatible versions
- One `.pkg` is not enough, also need `.pkg` for peripherals
- Introduce `.loads` files
 - Tells system where to find `.pkg`s
 - System pulls the `.pkg`s needed to upgrade all devices
 - Same mechanism as upgrades in the field
- Changes to dev workflow:
 - one or more build commands
 - single `binst` to update codec + peripherals:

binsting with .loads files

build
build



Changing binst

- Required additions to binst:
 - Prepare directory with .loads file, .sgn file and associated .pkg s
 - Serve this directory over HTTP
 - Tell codec to get .loads file from server, and wait for the required .pkgs to be pulled.
 - Retain fall-back to old push-method for older and non-functioning systems
- How to get there:
 - *First:* rewrite existing binst from Bash into Python
 - Then refactor code to ease readability and maintainability
 - Finally start adding new features
 - Generating .loads files
 - Knowledge of which peripheral .pkg s may be needed for what targets.
 - Generating signatures for .loads files
 - HTTP server to serve directory with .loads files and .pkg files

Rewriting binst from Shell to Python — Proper data structures

```
for target in "${targetarray[@]}"; do
    filter=
    finishcmd=
    file=
    echo "Installing ${target}"
    case "${target}" in
        asterix)
            desc="Asterix Complete Image, target arm linux"
            ;;
        asterix.apps)
            buildtarget="asterix /asterix/a8/apps"
            desc="Asterix arm-a8 application code, target arm linux"
            targetpath=/mnt/base/active/apps.img
            postbinstcmd="/bin/mainrestart update"
            ;;
        asterix.gui)
            buildtarget="asterix.gui"
            desc="Asterix GUI code, target arm linux"
            finaltargetpath=/mnt/base/active/gui.img
            targetpath="$finaltargetpath.tmp"
            postbinstcmd="/etc/init.d/S13gui unmount_img > /dev/null
                         mv $targetpath $finaltargetpath
                         /etc/init.d/S13gui mount_img > /dev/null 2>
                         ..."
            ;;
        ...
    esac
done
```

```
TARGETS = {
    'asterix': {
        'desc': 'Asterix complete image, target arm linux',
    },
    'asterix.apps': {
        'desc': 'Asterix arm-a8 application code, target arm linux',
        'subtarget': '/asterix/a8/apps',
        'destpath': '/mnt/base/active/apps.img',
        'posthook': '/bin/mainrestart update',
    },
    'asterix.gui': {
        'desc': 'Asterix GUI code, target arm linux',
        'buildtarget': 'asterix.gui',
        'destpath': '/mnt/base/active/gui.img.tmp',
        'posthook': '''
            /etc/init.d/S13gui unmount_img > /dev/null
            mv "$destpath" "${destpath%.tmp}" &&
            /etc/init.d/S13gui mount_img > /dev/null 2>
        ''',
    },
    ...
}
```

Rewriting binst from Shell to Python — Command-line interface

```
show_prods() {
    cat <<EOF
...
usage() {
    cat <<EOF
...
while [ -n "$1" ]; do
    case "$1" in
        --tlist|--list-targets)
            listtargets=1
            ;;
        -h|--help)
            help=1
            ;;
        -f|--file)
            shift
            if [ -z "$1" ]; then
                fail=1
                help=1
            fi
            userfile="$1"
            ;;
        ...
    done
```

```
def parse_args(*args):
    parser = argparse.ArgumentParser(
        usage=USAGE, description=sys.modules[__name__].__doc__)

    class ListTargets(argparse.Action):
        def __init__(self, default=argparse.SUPPRESS, **kwargs):
            super().__init__(nargs=0, default=default, **kwargs)

        def __call__(self, parser, *args):
            print('\n'.join(sorted(TARGETS.keys())))
            parser.exit()

    parser.add_argument(
        '--list-targets', '--tlist', action=ListTargets,
        help='List available targets.')

    # Allow positional <target> to alternatively be specified with -t/--target
    target_spec = parser.add_mutually_exclusive_group(required=True)
    target_spec.add_argument(
        'target', nargs='?', choices=sorted(TARGETS.keys()),
        metavar='<target>', help='Install image for this build target.')
    target_spec.add_argument(
        '--target', '-t', dest='target_alt', choices=sorted(TARGETS.keys())
        metavar='<target>', help='Install image for this build target.')
    ...
    parser.exit()
```

binst rewritten to Python

Stats

- 366 lines of Python (~19% reduction)
- Command-line interface (<20%)
- Data structure (~33%)

binst rewritten to Python

Stats

- 366 lines of Python (~19% reduction)
- Command-line interface (<20%)
- Data structure (~33%)

Next:

- Add `.loads` file generation
- Add signature generation
- Add HTTP server serving directory with the above + `.pkg` files

loadsfile.py

244 lines of Python, imported by binst, but also stand-alone:

```
$ ./loadsfile.py -h
usage: loadsfile.py [-h]
                     [--target {asterix,asterix.nocrypto,carbon,drishti,tempo,sunrise,zenith,halley,moody,...}]
                     [--file FILE] [--output OUTPUT] [--base-url BASE_URL]
                     [--pkgextract PKGEXTRACT] [--verify]
```

Utility for building .loads files around our build products. .loads file are JSON documents that declare a collection of related software images (.pkg files) for one **or** more of our products. .loads files are an important part of how we distribute software upgrades for our products. See <https://rdwiki.cisco.com/wiki/Swupgrade> for more details.

optional arguments:

-h, --help	show this help message and exit
--target {asterix,asterix.nocrypto,carbon,drishti,tempo,sunrise,zenith,halley,moody,pyramid,idefix}	Target to include in generated .loads file
--file FILE, -f FILE	PKG path corresponding to each given --target option
--output OUTPUT, -o OUTPUT	Where to write the resulting .loads file (default: stdout)
--base-url BASE_URL	Prefix to PKG filenames in resulting .loads file (default: "")
--pkgextract PKGEXTRACT	Path to pkgextract binary (default: find in \$PATH)
--verify	Do not reuse existing .pkg.loads files.

loadssign.py

295 lines of Python, imported by binst, but also stand-alone:

```
$ ./loadssign.py -h  
usage: loadssign.py [-h] [--release] [--test] {ticket,sign,verify} ...
```

Utility for signing .loads files with local test **keys** or SWIMS release **keys**.
.loads files must be signed in order for their contents to be trustworthy.
This is an important part of how we distribute software upgrades for our
products. See <https://rdwiki.cisco.com/wiki/Swupgrade> for more details.

positional arguments:

{ticket,sign,verify}	
ticket	Create a SWIMS ticket for signing release S/W
sign	Generate a .loads signature
verify	Verify a .loads signature

optional arguments:

-h, --help	show this help message and exit
--release	Sign/verify with key/certificate for release S/W
--test	Sign/verify with key/certificate for test S/W

loadsdir.py

519 lines of Python, imported by binst, but also stand-alone:

```
$ ./loadsdir.py -h
usage: loadsdir.py [-h]
                   [--target {asterix,asterix.nocrypto,carbon,drishti,tempo,sunrise,zenith,halley,moodys,...}]
                   [--file FILE] [--version VERSION] [--loads-name LOADS_NAME]
                   [--symlink] [--copy] [--deps] [--objdir OBJDIR]
                   [--validate] [--ticket TICKET] [--serve]
destination
```

Utility for building loads directories around our build products. loads dirs are directories containing a .loads file and all the PKG files referenced from the .loads file. See <https://rdwiki.cisco.com/wiki/Swupgrade> for more details.

positional arguments:

destination Where to write the loads directory contents

optional arguments:

[...]	
--symlink	Symlink PKG files into loads dir (this is the default behavior)
--copy	Copy PKG files into loads dir (instead of symlinking)
--deps, -d	Automatically include dependencies of the given target
--objdir OBJDIR, -O OBJDIR	Look here for dependents' build output (default: _build)
--validate	Validate any .loads and .pkg files found within the loads dir.
--ticket TICKET	Use this SWIMS ticket to verify .loads signatures.
--serve	Serve loads dir over HTTP until you press Ctrl+C.

Integrating HTTP server into binst.py

```
if args.loads:
    assert args.target.support_loads()
    server = LoadsServer(args.target, image_path, args.objdir)
    script = '; '.join([
        'origin=$(echo $SSH_CLIENT | cut -d" " -f1)',
        'upgrade_url="http://$origin:{0.port}/{0.loadspath}"'.format(server),
        'echo "xcom SystemUnit SoftwareUpgrade URL: $upgrade_url" | tsh',
    ])
    ssh_cmd = build_ssh_cmd(
        remote_user,
        ssh_address(args.destination),
        script,
        ssh=args.target.ssh)
    print('Triggering {} to upgrade from our port {}...'.format(
        args.destination, server.port))
    if args.verbose:
        print('Running: {}'.format(ssh_cmd))
    if subprocess.call(ssh_cmd, shell=True) != 0:
        print('Failed to trigger upgrade (command: {}).'.format(ssh_cmd))
    else: # Hand control over to loads server.
        if server.serve(): # Files were served to destination. We're done.
            return 0
        else: # Destination failed to request anything from us.
            print('No upgrade requests from {}!'.format(args.destination))
    server.cleanup()
print('Falling back to old/--no-loads behavior...')
```

That's all folks!



Summary

- Use shell scripts for the simplest tasks (3 lines of shell is simpler than 7 lines of python)
- Switch to Python when things start growing
 - Complex logic
 - Data structures
 - Modularity/reuse
 - Configurability
- Write Python scripts that can *both* be imported and run from the command-line
- Refactor mercilessly
- Unit tests gives you confidence to refactor!
- Use the Python standard library:
 - `argparse`
 - `subprocess`
 - `pathlib`

Summary

- Use shell scripts for the simplest tasks (3 lines of shell is simpler than 7 lines of python)
- Switch to Python when things start growing
 - Complex logic
 - Data structures
 - Modularity/reuse
 - Configurability
- Write Python scripts that can *both* be imported and run from the command-line
- Refactor mercilessly
- Unit tests gives you confidence to refactor!
- Use the Python standard library:
 - `argparse`
 - `subprocess`
 - `pathlib`

«This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.»

— Doug McIlroy

Thanks!

Questions?

Johan Herland

@jherland

<jherland@cisco.com>

<johan@herland.net>

Slides on [GitHub](#).

Slideshow created with [remark](#).