CS615 - Aspects of System Administration

Writing System Tools

Department of Computer Science
Stevens Institute of Technology
Jan Schaumann
jschauma@stevens.edu
https://www.cs.stevens.edu/~jschauma/615/

What should be automated?

- software installation / upgrades / audits
- account creation / deletion
- system configuration changes
- log and event processing

...and anything / everything else in between!

Words of Wisdom

Anything you do more than once is worth automating.

Why automate anything?

- we're lazy
- we're unreliable and make mistakes
- we're forgetful

Whom do we automate things for?

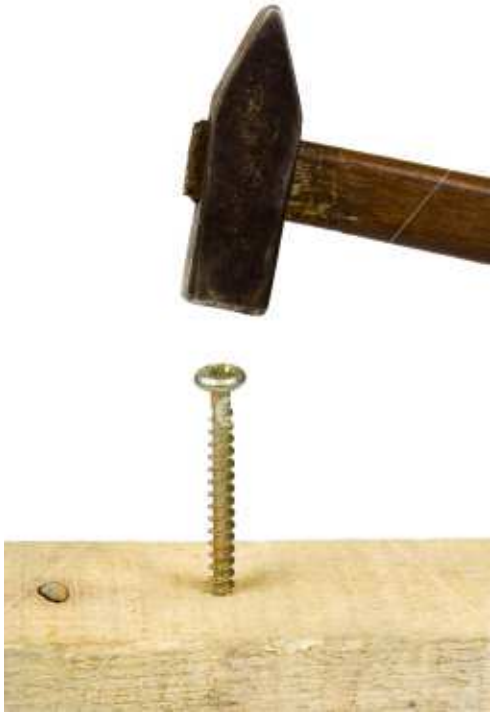
- ourselves
- our peers
- our users
- anybody else

Tools

The right tool?

The right tool?

The right tool?



The right tool?

Bourne shell (/bin/sh)

- lowest common denominator
- available and reliable on most platforms (but beware of non-portable bash(1) ”enhancements”)
- beware of ”quick-and-dirty” solutions, they grow to become unmaintainable
- treat shell as any other programming language:
  - use functions
  - use suitably scoped variables
  - follow Unix philosophy
  - properly package your tool

The right tool?

Perl, Python, Ruby, Node, ...

- suitable for moderately complex tasks
- move to these when sed(1), awk(1), etc. become too cumbersome
- text manipulation frequently easier
- beware of ”quick-and-dirty” solutions, they grow to become unmaintainable
- try to build self-contained modules that can be tested independent of the ”main” program
- wealth of libraries available – use them! (And remember to explicitly require them.)
- properly package your tool

The right tool?

Perl, PHP, Tcl, JavaScript, CoffeeScript, ...

- http/web server interfaces
- CGI ”scripts” / server-side execution
- interface with/utilize APIs in a specific domain/vendor products
- frequent cause of all sorts of security problems due to interface with user data / exposure on the internet

The right tool?

Java, Scala, Clojure, Rhino, ...

- know your primary applications
- interface with / extend / tie into JVM

The right tool?

C, C++, Go, Rust, ...

- performance benefits
- portability
- sufficient low-levelness
- systems understanding
- fix/patch your other tools / the system

Interpreted Languages

General advantages:

- short development cycle
- normally facilitate things like string manipulation, arithmetic and more complex regular expressions
- easily handle multiple file handles and other I/O
- some security features
- tens of thousands of special- and general-purpose modules available

Interpreted Languages

General Disadvantages:

- no one tool fits all purposes
- tens of thousands of special- and general-purpose modules available
  $=>$ lots of duplication, stale code, questionable quality
  packaging and dependency resolution nightmares
- security features frequently neglected or circumvented (”too hard” or more precisely ”inconvenient”)
- everybody has their particular favorite (and dislikes one or the other)
- interpreter not (necessarily) universally available / installed

Approaching Automation

Three basic categories:

- scripting
- programming
- software engineering

Approaching Automation

Three basic categories:

- scripting

Approaching Automation

Three basic categories:

- scripting
  - automating *very* simple tasks

Approaching Automation

Three basic categories:

- scripting
  - automating *very* simple tasks
  - customization of user environment

Approaching Automation

Three basic categories:

- scripting
  - automating *very* simple tasks
  - customization of user environment
  - often only suitable for one individual user

Approaching Automation

Three basic categories:

- scripting
  - automating *very* simple tasks
  - customization of user environment
  - often only suitable for one individual user
  - usually eventually evolves into larger programs

Approaching Automation

Three basic categories:

- programming

Approaching Automation

Three basic categories:

- programming
  - suitable for simple to moderately complex tasks

Approaching Automation

Three basic categories:

- programming
  - suitable for simple to moderately complex tasks
  - results frequently used by a small base of users

Approaching Automation

Three basic categories:

- programming
  - suitable for simple to moderately complex tasks
  - results frequently used by a small base of users
  - uses basic framework or common toolkits

Approaching Automation

Three basic categories:

- programming
  - suitable for simple to moderately complex tasks
  - results frequently used by a small base of users
  - uses basic framework or common toolkits
  - provides consistent interface

Approaching Automation

Three basic categories:

- programming
  - suitable for simple to moderately complex tasks
  - results frequently used by a small base of users
  - uses basic framework or common toolkits
  - provides consistent interface
  - may evolve into full product

Approaching Automation

Three basic categories:

- software development

Approaching Automation

Three basic categories:

- software development
  - required for any reasonably complex task

Approaching Automation

Three basic categories:

- software development
  - required for any reasonably complex task
  - uses formal software engineering approach (measurable goals, requirements, specifications, ...)

Approaching Automation

Three basic categories:

- software development
  - required for any reasonably complex task
  - uses formal software engineering approach (measurable goals, requirements, specifications, ...)
  - may evolve from previous prototypes

Approaching Automation

Three basic categories:

- software development
  - required for any reasonably complex task
  - uses formal software engineering approach (measurable goals, requirements, specifications, ...)
  - may evolve from previous prototypes
  - requires ongoing continous maintenance / development efforts

Where are you?

Make sure to understand your requirements:

- motivation / goals
- target audience
- scope
- dependencies

Regular Expressions

Example exercises:

- check if a string is a valid date
  `03/07/2016, 7.3.2016, 2016-03-07, 2016/03/07, ...`
- check if a string is a valid IPv4 address
  `0.0.0.0, 255.255.255.255, ...`
- check if a string is a valid IPv6 address
  `::1, fe80::e276:63ff:fe72:3900%xennet0, 2001:470:30:84:e276:63ff:fe72:3900, ...`
- extract all URLs from a document
  `http://example.com, ftp://example.com/dir/file.html,`
  `https://example.com/?foo=bar&blob=';alert(1);,`
  `http://example.com/?redir=http://example.com, ...`
- extract all proper words from a document
  `word, it's, Name, Henry the 3rd, cross-site scripting, ...`

Regular Expressions

IPv4:

Regular Expressions

IPv4:

```
(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?
[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?\.(25[0-5]|2[0-4][0-9]|[01]
?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)
```

Regular Expressions

IPv4:

```
(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?
[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?\.(25[0-5]|2[0-4][0-9]|[01]
?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)
```

IPv6:

```
(([0-9a-fA-F]{1,4}:){7,7}[0-9a-fA-F]{1,4}|([0-9a-fA-F]{1,4}:){1,7}:|
([0-9a-fA-F]{1,4}:){1,6}:[0-9a-fA-F]{1,4}|([0-9a-fA-F]{1,4}:){1,5}(:
[0-9a-fA-F]{1,4}){1,2}|([0-9a-fA-F]{1,4}:){1,4}(:[0-9a-fA-F]{1,4}){1
,3}|([0-9a-fA-F]{1,4}:){1,3}(:[0-9a-fA-F]{1,4}){1,4}|([0-9a-fA-F]{1,
4}:){1,2}(:[0-9a-fA-F]{1,4}){1,5}|[0-9a-fA-F]{1,4}:((:[0-9a-fA-F]{1,
4}){1,6})|:((:[0-9a-fA-F]{1,4}){1,7}|:)|fe80:(:[0-9a-fA-F]{0,4}){0,4
}%[0-9a-zA-Z]{1,}|::(ffff(:0{1,4}){0,1}:){0,1}((25[0-5]|(2[0-4]|1{0,
1}[0-9]){0,1}[0-9])\.){3,3}(25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])|
([0-9a-fA-F]{1,4}:){1,4}:((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])\.
){3,3}(25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]))
```

Regular Expressions

Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems.

Better:

```
if (inet_pton(AF_INET, $ip)) {
        # AF_INET
} elsif (inet_pton(AF_INET6, $ip)) {
        # AF_INET6
} else {
        # not an IP address
}
```

Know when you need to be precise, and when 'good enough' is good enough.

Hooray!

5 Minute Break

User Interface

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.

Know your languages / eco-system

Some advice transcends language:

```
$ echo import this | python
```

The Zen of Python

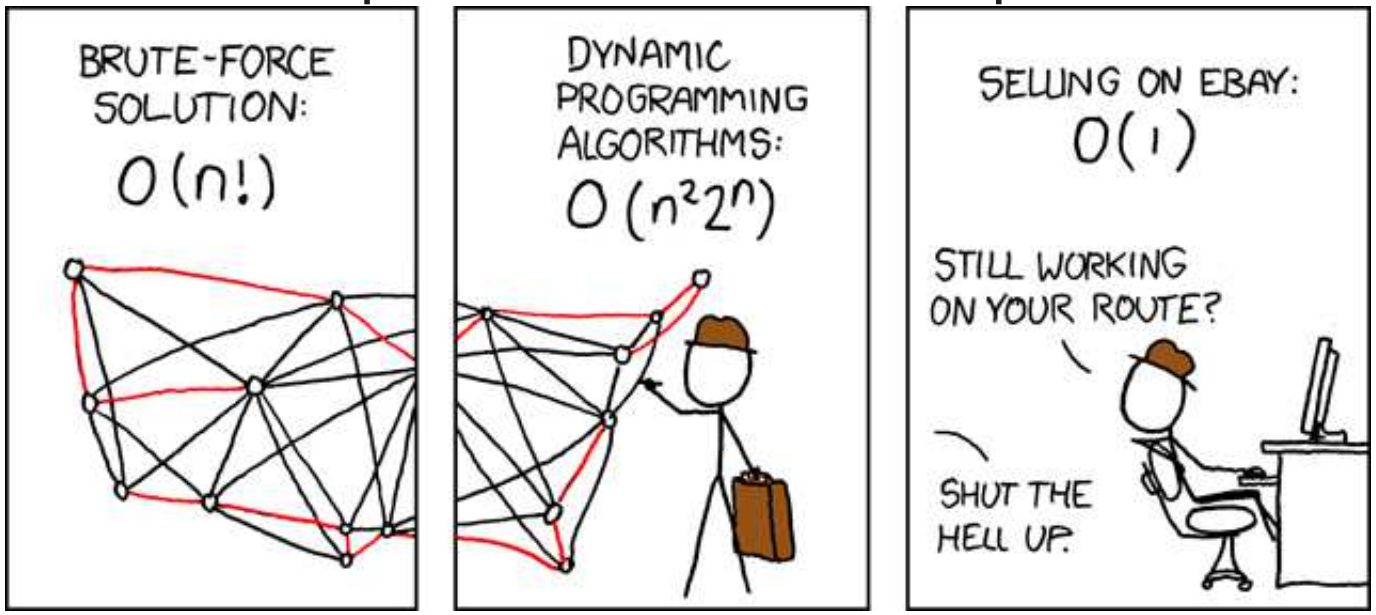
Beautiful is better than ugly.

The Zen of Python

Explicit is better than implicit.

The Zen of Python

Simple is better than complex.



http://xkcd.com/399/

The Zen of Python

Complex is better than complicated.

The Zen of Python

Flat is better than nested.
Sparse is better than dense.

Readability counts.

The Zen of Python

Special cases aren't special enough to break the rules.

The Zen of Python

Special cases aren't special enough to break the rules.

Although practicality beats purity.

The Zen of Python

Errors should never pass silently.

The Zen of Python

Errors should never pass silently.

(That would be implicitly accepted failure.)

The Zen of Python

Errors should never pass silently.

(That would be implicitly accepted failure.)

(You know what would be better than something *implicit*?)

The Zen of Python

Errors should never pass silently.

(That would be implicitly accepted failure.)

(You know what would be better than something *implicit*?)

(Why, of course, something *explicit*!)

The Zen of Python

Errors should never pass silently.

Unless explicitly silenced.

The Zen of Python

In the face of ambiguity, refuse the temptation to guess.

The Zen of Python

There should be one – and preferably only one –
obvious way to do it.

The Zen of Python

There should be one – and preferably only one – obvious way to do it.

Although that way may not be obvious at first unless you're Dutch.

The Zen of Python

Now is better than never.

The Zen of Python

Now is better than never.

Although never is often better than *right* now.

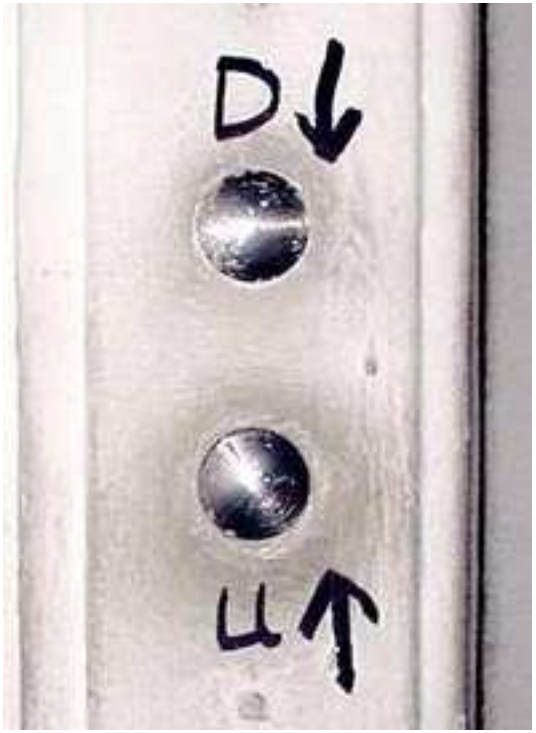
The Zen of Python

If the implementation is hard to explain, it's a bad idea.

The Zen of Python

If the implementation is easy to explain,
it *may* be a good idea.

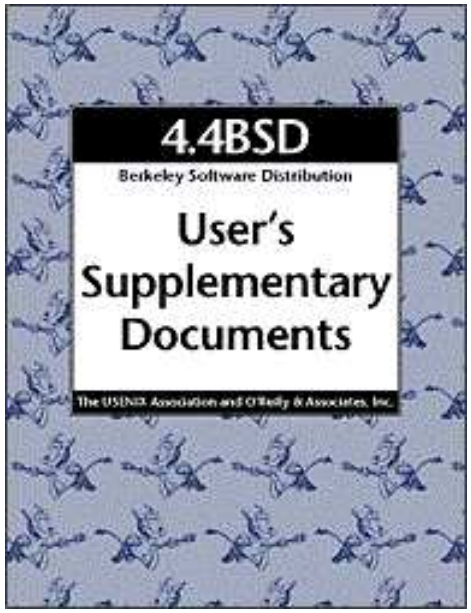
A simple interface, easy to explain. Yet...

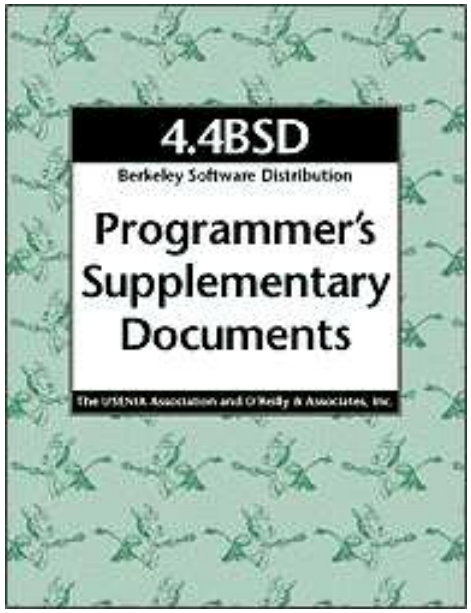


The Zen of Python

Namespaces are one honking great idea – let's do more of those!

Documentation





WTFM

Robustness Principle or Postel's Law

Be conservative in what you do; be liberal in what you accept from others.

POLA

Principle of Least Astonishment

Know your Users

Whom do we automate things for?

- ourselves
- our peers
- our ”users”
- anybody else

Avoid the Quick Fix

There's nothing as permanent as a temporary solution.

Avoid the Project That Was Never Finished

Don’t let the Perfect be the enemy of the Good.

Avoid Feature Creep

http://www.feepingcreatures.com

Release Early, Release Often

"More users find more bugs."

F. Brooks, "The Mythical Man Month"

Take a good look in the mirror!

Looks like *you* are the ass.

Learn to write a detailed bug report

Pre-requisite:

- RTFM
- Internet Search
- Know Your Community

Learn to write a detailed bug report

Pre-requisite: Do your homework.

Required:

- Description Of Problem
- Steps To Reproduce
- Expected Results
- Actual Results

Learn to write a detailed bug report

Pre-requisite: Do your homework.

Required:

- Description Of Problem
- Steps To Reproduce
- Expected Results
- Actual Results

Optional / recommended:

- Screenshots / *exact* copy of terminal I/O (`script(1)`)
- Suggested Remediation
- Code Patch

Increase the Bus Factor

“Just friends.”

Collaboration is non-optional

Efficient use of Version Control Systems is a requirement. They allow you to:

- collaborate with others
- simultaneously work on a code base
- keep old versions of files
- keep a log of the who, when, what, and why of any changes
- perform release engineering by creating *branches*

Commit Messages

Commit messages are like comments: often useless and misleading, but critical in understanding human thinking behind the code.

Commit messages are full sentences in correct and properly formatted English.

Commit messages briefly summarize the *what*, but provide important historical context as to the *how* and, more importantly, *why*.

Commit messages SHOULD reference and integrate with ticket tracking systems.

See also:

- `http://is.gd/Wd1LhA`
- `http://is.gd/CUtwhA`
- `http://is.gd/rPQj5E`

Fix Broken Windows

Scalability

- Simplify!

Scalability

- Simplify!
- Reduce or eliminate interactions with the user.

Scalability

- Simplify!
- Reduce or eliminate interactions with the user.
- Premature optimization is the root of all evil.

Scalability

- Simplify!
- Reduce or eliminate interactions with the user.
- Premature optimization is the root of all evil.
- So is excusing shoddy programming.

Scalability

- Simplify!
- Reduce or eliminate interactions with the user.
- Premature optimization is the root of all evil.
- So is excusing shoddy programming.
- Fix *all* warnings and errors.

Scalability

- Simplify!
- Reduce or eliminate interactions with the user.
- Premature optimization is the root of all evil.
- So is excusing shoddy programming.
- Fix *all* warnings and errors.
- Document all assumptions. Be specific.

Scalability

- Simplify!
- Reduce or eliminate interactions with the user.
- Premature optimization is the root of all evil.
- So is excusing shoddy programming.
- Fix *all* warnings and errors.
- Document all assumptions. Be specific.
- Always apply the Principle of Least Privilege.

Scalability

- Simplify!
- Reduce or eliminate interactions with the user.
- Premature optimization is the root of all evil.
- So is excusing shoddy programming.
- Fix *all* warnings and errors.
- Document all assumptions. Be specific.
- Always apply the Principle of Least Privilege.
- Assume hostile input and usage.

Scalability

- Simplify!
- Reduce or eliminate interactions with the user.
- Premature optimization is the root of all evil.
- So is excusing shoddy programming.
- Fix *all* warnings and errors.
- Document all assumptions. Be specific.
- Always apply the Principle of Least Privilege.
- Assume hostile input and usage.
- Understand your code.

Program Maintenance

"... is an entropy-increasing process, and even its most skillful execution only delays the subsidence of the system into unfixable obsolescence."

F. Brooks, "The Mythical Man Month"

Toss it!

HW

https://www.cs.stevens.edu/~jschauma/615/s17-hw5.html
https://www.cs.stevens.edu/~jschauma/615/s17-hw6.html

Reading

Shell:

- `http://www.tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html`
- `http://www.tldp.org/LDP/abs/html/`
- `http://sed.sourceforge.net/sed1line.txt`
- `csh(1)`, `ksh(1)`, `sh(1)`

Perl:

- `http://www.perl.com`
- `http://www.cpan.org`
- `perl(1)`, `perldoc(1)`, `perlfaq(1)`

Python:

- `http://www.python.org`
- pydoc

Reading

Ruby:

- http://www.ruby-lang.org/en/
- http://is.gd/cE7iFR
- http://is.gd/DR4aNU

Other:

- http://www.regex.alf.nu/
- http://is.gd/jDDGpW
- https://www.netmeister.org/blog/writing-tools.html