

Scripting languages

- **originally tools for quick hacks, rapid prototyping, gluing together other programs, ...**
- **evolved into mainstream programming tools**
- **characteristics**
 - strings as basic (or only) data type
 - associative arrays as a basic aggregate type
 - regular expressions (maybe) built in
 - minimal use of types, declarations, etc.
 - usually interpreted instead of compiled
 - easy to get started with
- **examples**
 - **shell**
 - **Awk**
 - **Perl, PHP, Ruby**
 - **Python**
 - **Tcl**
 - **Javascript**
 - **VBScript, JScript**
 - ...

Shells and shell programming

- **shell: a program that helps run other programs**
 - intermediary between user and operating system
 - basic scripting language
 - programming with programs as building blocks
- **an ordinary program, not part of the system**
 - it can be replaced by one you like better
 - therefore there are lots of shells, reflecting history and preferences
- **popular shells:**
 - **sh** Bourne shell (Steve Bourne, Bell Labs -> ...) emphasizes running programs and programmability syntax derived from Algol 68
 - **csch** C shell (Bill Joy, UC Berkeley -> Sun) interaction: history, job control, command & filename completion, aliases more C-like syntax not as good for programming (at least historically)
 - **ksh** Korn shell (Dave Korn, Bell Labs -> AT&T Labs) combines programmability and interaction syntactically, superset of Bourne sh provides all csch interactive features + lots more
 - **bash** GNU shell mostly ksh + much of csch
 - **tcsh** evolution of csch

Features common to Unix shells

- **command execution**
 - + built-in commands, e.g., cd
- **filename expansion**
 - * ? [...]
- **quoting**
 - rm '*' Careful!!!
 - echo "It's now `date`"
- **variables, environment**
 - PATH=/bin:/usr/bin in ksh & bash
 - setenv PATH /bin:/usr/bin in (t)csH
- **input/output redirection, pipes**
 - prog <in >out, prog >>out
 - who | wc
 - slow.1 | slow.2 & *asynchronous operation*
- **executing commands from a file**
 - arguments can be passed to a shell file (\$0, \$1, etc.)
 - if made executable, indistinguishable from compiled programs

provided by the shell, not each program

Shell programming

- **the shell is a programming language**
 - the earliest scripting language
- **string-valued variables**
- **limited regexps** mostly for filename expansion
- **control flow**
 - if-else
 - if cmd; then cmds; elif cmds; else cmds; fi (sh...)
 - if (expr) cmds; else if (expr) cmds; else cmds; endif (csh)
 - while, for
 - for var in list; do commands; done (sh, ksh, bash)
 - foreach var (list) commands; end (csh, tcsh)
 - switch, case, break, continue, ...
- **operators are programs**
 - programs return status
 - 0 == success, non-0 == various failures
- **shell programming out of favor**
 - graphical interfaces
 - scripting languages
 - e.g., system administration
 - setting paths, filenames, parameters, etc
 - now often in Perl, Python, PHP

bundle: making "shell archives"

Use:

```
$ bundle foo bar >bundle.out
  combines text files "foo" and "bar" into a shell file
  that recreates foo and bar when it is executed.
```

Implementation:

```
echo '# To unbundle, sh this file'
for i in $*
do echo "echo $i 1>&2"
  echo "sed 's/-//>'>$i <<'End of $i'"
  sed 's/^/-/' $i
  echo "End of $i"
done
```

Output:

```
# To unbundle, sh this file
echo foo 1>&2
sed 's/-//>'>foo <<'End of foo'
-contents of foo...
End of foo
echo bar 1>&2
sed 's/-//>'>bar <<'End of bar'
-contents of bar...
End of bar
```

To unbundle:

```
$ sh bundle.out
```

How big should a program be?

```
$ wc bundle
   7   29  156 bundle
$ wc shar.c
2130 6659 53377 shar.c
```

"Shar puts readable text files together in a package from which they are easy to extract. The original version was a shell script posted to the net, shown below:

```
#Date: Mon Oct 18 11:08:34 1982
#From: decvax!microsof!uw-beave!jim
      (James Gosling at CMU)

AR=$1
shift
for i do
  echo a - $i
  echo "echo x - $i" >>$AR
  echo "cat >$i <<'!Funky!Stuff!'" >>$AR
  cat $i >>$AR
  echo "!Funky!Stuff!" >>$AR
done
```

I rewrote this version in C to provide better diagnostics and to run faster. ..."

Shell programming

- **shell programs are good for personal tools**
 - tailoring environment
 - abbreviating common operations
(aliases do the same)
- **gluing together existing programs into new ones**
- **prototyping**
- **sometimes for production use**
 - e.g., configuration scripts

- **But:**
 - shell is poor at arithmetic, editing
 - macro processing is a mess
 - quoting is a mess
 - sometimes too slow
 - can't get at some things that are really necessary

- **this leads to scripting languages**