Unix Philosophy and Culture

- Unix Philosophy (Early Originators)
- Unix Philosophy (Eric S. Raymond)
- Tenets (Gancarz)
- Lesser Tenets (Gancarz)
- Unix Culture
- References

Unix Philosophy (Early Originators)

- *Doug MacIlroy*

  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.

- *Rob Pike*

  1. You can't tell where a program is going to spend its time. Bottlenecks occur in surprising places, so don't try to second guess and put in a speed hack until you've proven that's where the bottleneck is.

  2. Measure. Don't tune for speed until you've measured, and even then don't unless one part of the code overwhelms the rest.

  3. Fancy algorithms are slow when *n* is small, and *n* is usually small. Fancy algorithms have big constants. Until you know that *n* is

frequently going to be big, don't get fancy. (Even if *n* does get big, use Rule 2 first)

4. Fancy algorithms are buggier than simple ones, and they're much harder to implement. Use simple algorithms as well as simple data structures.

5. Data dominates. If you've chosen the right data structures and organized things well, the algorithms will almost always be self-evident. Data structures, not algorithms, are central to programming.

6. There is no Rule 6.

- *Ken Thompson*

  When in doubt, use brute force.

Unix Philosophy (Eric S. Raymond)

- Write small pieces connected by clean interfaces.
- Design programs to communicate easily with other programs.
- Robustness is the child of transparency and simplicity.
- Design for simplicity; add complexity only where you must.
- Design for transparency; spend effort early to save effort later.
- In interface design, obey the *Rule of Least Surprise*.
- Programmer time is expensive; conserve it in preference to machine time.

- Avoid hand-hacking; write programs to write programs when you can.
- Use smart data so program logic can be stupid and robust.
- Prototype, then polish. Get it working before you optimize it.
- Distrust all claims for *one true way*.

Tenets (Gancarz)

- Small is beautiful.
- Make each program do one thing well.
- Prototype as soon as possible.
- Choose portability over efficiency.
- Store numerical data in flat ASCII files.
- Use software leverage to your advantage.
- Use shell scripts to increase leverage and portability.
- Avoid captive user interfaces.
- Make every program a filter.

Lesser Tenets (Gancarz)

- Allow the user to tailor the environment.
- Make OS kernels small and lightweight.
- Use lower case and keep it short.
- Save trees.
- Silence is golden.
- Think parallel.
- The sum of the parts is greater than the whole.
- Look for the 90% solution.
- Worse is better.

Unix Culture

- Sharing of source code (largely true on most computer systems before 1980's).
- Open. Read access across entire system. On many early systems, every one had `root` access.
- Academic.

References

Richard P. Gabriel, *Worse Is Better* (and related essays), links from http://www.dreamsongs.com/WorseIsBetter.html.

Mike Gancarz, *The Unix Philosophy*, Digital Press, 1994.

Mike Meyer, *Good enough is best*, at http://mike.mired.org/writing/good-enough.

Rob Pike, *Notes on Programming in C*, at http://www.lysator.liu.se/c/pikestyle.html.

Eric. S. Raymond, *The Art of Unix Programming*, Addison-Wesley, 2003. At http://www.faqs.org/docs/artu/.

Wikipedia, *Unix Philosophy*, at http://en.wikipedia.org/wiki/Unix_philosophy.

Rob Pike, Brian Kernighan, *Program Design in the Unix Environment*, at http://harmful.cat-v.org/cat-v/unix_prog_design.pdf.