

Unleashing the Shell

Hands-On UNIX System Administration DeCal

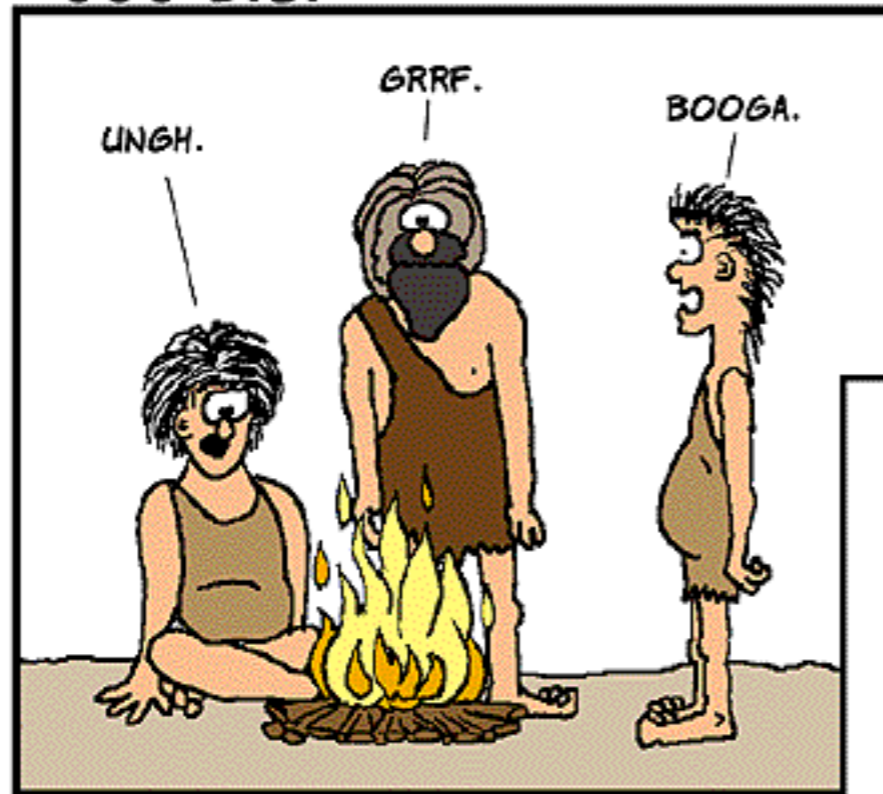
Week 6 — 28 February 2011

Last time

- Compiling software and the three-step procedure (`./configure && make && make install`).
- Dependency hell and package managers.
- Lab assignment: compiling nethack. (If you're not yet in a final project group, ask around — some groups are a few short.)

EVOLUTION OF LANGUAGE THROUGH THE AGES.

6000 B.C.



2000 A.D.



COPYRIGHT (C) 1999 ILLIAD

[HTTP://WWW.USERFRIENDLY.ORG/](http://www.userfriendly.org/)

source: <http://ars.userfriendly.org/cartoons/?id=19990815>

Grok awk?

- **grep:** match input based on a pattern (or regular expression — more on that later). You've already used it, but today's lecture will let you unlock its full potential.
- **awk** and **sed:** powerful programming languages designed for text processing. We'll be using awk for field extraction and sed for regex find-and-replace searches.

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.

OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!



BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!



IT'S HOPELESS!

EVERYBODY STAND BACK.



I KNOW REGULAR EXPRESSIONS.



source: <http://xkcd.com/208/>

Classical Regular Expressions

- Regular expressions denote formal languages, which are sets of strings (of symbols from some alphabet).
- Appropriate since internal structure not all that complex yet.
- Expression R denotes language $L(R)$:
 - $L(\epsilon) = L("") = \{""\}$.
 - If c is a character, $L(c) = \{ "c" \}$.
 - If R_1, R_2 are r.e.s, $L(R_1R_2) = \{x_1x_2 \mid x_1 \in L(R_1), x_2 \in L(R_2)\}$.
 - $L(R_1|R_2) = L(R_1) \cup L(R_2)$.
 - $L(R^*) = L(\epsilon) \cup L(R) \cup L(RR) \cup \dots$.
 - $L((R)) = L(R)$.
- Precedence is '*' (highest), concatenation, union (lowest). Parentheses also provide grouping.

Regular expressions

- ... don't worry, you don't need to understand set theory to use regexes!
- Syntax and features vary from program to program (consult documentation to see what exactly you can do), but these basics are universal.

Regular expressions

- Most characters match themselves ("cat" matches "cat," "bobcats," "catastrophe"...).
- **[a-z]** is a *character class* that matches one character from the specified set. **[^a-z]** matches one character *not* in the set.
- **.** (dot) matches any character.
- **^** and **\$** match start and end of line.

Regular expressions

- * matches the preceding symbol any number of times, + at least one time, and ? at most one time.
- () (parentheses) group symbols and | (pipe) separates alternatives ("hat|cat").
- \1, \2, \3... refer to the *n*th grouped subexpression. "(cat)\1" matches "catcat".

Examples...

- Experiment! Any good editor will have regular expression support (or use grep).
- `^$` matches an empty line. Hint: `grep -v`.
- `#.*$` matches everything from a hash mark to end-of-line (config file comment).
- `[a-z]+@[a-z]+\.(org|net|com)` naively matches email addresses. To do it right...

Examples...

- Backreferences (\1, \2, \3...) are most useful when doing regex replacements.

Garcia, Dan

Harvey, Brian

Hilfinger, Paul

Sinclair, Alistair

Shewchuk, Jonathan

- What does `sed -E -e 's/([A-Za-z]+), ([A-Za-z]+)/\2 \1/g'` do to this file?

sed

- `sed -e 's/old/new/g'` replaces *old* with *new* globally. Add the `-E` flag for extended (modern) regular expressions.
- `sed -e 's/old/new/g' file > file` will clobber your file, not update it — be careful! To edit in-place, use `-i`.
- And there's more! RTFM for details.

awk

- `awk '{print $1}'` prints its input's first *field*. By default, fields are delimited by any number of spaces (change the field separator with the `-F` option).
- `ls -l /etc | awk '{print $NF " is owned by " $3}'` extracts the filename and owner fields.

Miscellany

- **cut** extracts sections of its input — you can select arbitrary bytes, characters, or fields (with whatever delimiter you like).
e.g., `getent passwd | cut -f1,5 -d:`
- **tr** deletes or replaces (translates) characters. Only uses stdin (not UUOC!).
e.g., `cat /etc/group | tr -d '\n'`
e.g., `echo "go bears" | tr a e`

Anatomy of a script

- A shell script is, at its simplest, a plain text file containing a list of commands.
- Scripts usually have *shebang lines* (e.g., `#!/bin/sh`) indicating what program to process them with, so they can be run as ordinary programs.
- Shell scripts can have variables, functions, control flow...

Variables

- **Assignment:** `VARNAME=value`.
Variables can be lowercase, but are usually uppercase. Can also use substitution, as in `EDITOR=`whereis vim``.
- **Reference:** `echo $VARNAME`.
Note that `echo '$VARNAME'` doesn't evaluate the variable.

Variables

- bash provides some variables to aid in shell scripting. Here are a few:
 - `$#` — number of arguments passed to your script. (`./script foo bar baz => 3.`)
 - `$0, $1, $2...` — arguments (`$0` is your script, like `argv[0]` in C's `main()` function).
 - `@$` — all arguments in one variable.

Functions

- Function declarations, by example:

```
defenestrate() {  
    echo "Throwing $@ out the  
        window."  
}  
defenestrate your homework
```

- Note that arguments are handled with special variables, not declared as in C.

Control flow

- **For loops** iterate over everything in a list. If you need to work with numbers, use `{0..100}` or ``seq 0 100``. Example:

```
for DOCTOR in {hartnell,troughton,pertwee,baker,
davison,colin,mccoy,mcgann,eccleston,tennant}; do
    mkdir -p /mnt/$DOCTOR
    mount -o loop /xen/domains/$DOCTOR/disk.img /mnt/
$DOCTOR
done
```

Control flow

- bash also supports **while loops**. E.g.:

```
while true; do
    sleep 100
    uptime      # includes load info
done
```

(If you actually want to do something like this, try the watch command.)

Control flow

- **If expressions** rely on a command called `test`, which is often abbreviated to `[`. There are *lots* of tests available — `test(1)` is well worth a read. Here's an example:

```
if [ $# -eq 0 ]; then
    echo "Usage:  $0 [args]"
    exit 1
fi
```

...and more!

Instead of deluging you with the details, I've just given you a high-level overview of shell shenanigans. If there's something specific you'd like to know how to do, please ask!