Thinking Outside the SAS Box: UNIX Tools

David B. Horvath, CCP, MS

PhilaSUG Winter Meeting

Drexel University, Dornsife School of Public Health

March 12, 2018

Thinking Outside the SAS Box: UNIX Tools

The Author can be contacted at:

504 Longbotham Drive, Aston PA 19014-2502, USA
Phone: 1-610-859-8826
Email: dhorvath@cobs.com
Web: http://www.cobs.com/
LI: http://www.linkedin.com/in/dbhorvath

Abstract

- What do you do when you get a file that is too large (too wide a record or too many records or both) to easily view? How do you know the CSV or TAB-delimited file really is structured correctly? When you get a fixed format file, how do you figure out what the field at position 468 really looks like (what data it contains)? How do you deal with files that do not read correctly, or worse, break the engine you’re using?
- There are a number of UNIX/Linux tools to help resolve those issues. Those same tools are also available under Windows with add-ons from Microsoft and other sources. They are included with the MAC OSX operating system!
- This session will review a few of these commands including live demos under Windows. These examples come from real life problem solving.

My Background

- David is an IT Professional (not a modeler) who has worked with various platforms since the 1980's with a variety of tools.
- He has presented at PhilaSUG previously and has presented workshops and seminars in Australia, France, the US, Canada, and Oxford England (about the British Author Nevil Shute) for various organizations including SESUG. He will be presenting at SGF 2018!
- He holds an undergraduate degree in Computer and Information Sciences from Temple University and a Masters in Organizational Dynamics from UPENN. He achieved the Certified Computing Professional designation with honors.
- Most of his career has been in consulting (although recently he has been in-house) in the Philadelphia PA area.
- He has several books to his credit (none directly SAS related) and is an Adjunct Instructor covering IT topics, University of Phoenix.
- He has worked with Base SAS on Mainframe, UNIX, and PC Platforms, primarily as an ETL tool or Programming Language.

Some Terminology

- GNU – GNU is Not UNIX
  - A command line interface environment that mimics those under UNIX
  - Started in 1983 by Richard M. Stallman at MIT
- FOSS – Free/Open Source Software
  - "FOSS programs are those that have licenses that allow users to freely run the program for any purpose, modify the program as they want, and also to freely distribute copies of either the original version or their own modified version. " – http://www.webopedia.com/TERM/F/FOSS.html
- Open System
  - Generically: Any system that follows a public set of standards
  - In this case: A system that meets the POSIX/UNIX standards

Some Terminology

- UNIX
  - Registered trademark of The Open Group
  - Often used as generic term as in "UNIX-like"
  - An Open System
  - But not FOSS – a mistake my students often make
  - Often referred to as "flavors" (i.e., Solaris, AIX, Tru64, etc.)

- Linux
  - A kernel plus the GNU command environment
  - An Open System **and** FOSS
  - Referred to as "distros" or distributions (i.e., Red Hat, Ubuntu, Knoppix)

- Cygwin
  - Open Source collection of GNU tools that runs under Windows

Right Tool, Right Place

- Where does the file come from?
  - Would it be better to process it there?
  - Rather than wait for transmission to another platform
  - Is that system better prepared to process it?

- Are there better tools than you're using now?
  - Do you always need to write code?
  - Value of cross-platform applicability

- Your alternatives
  - Microsoft Subsystem for UNIX-based Applications (SUA)
  - Windows Subsystem for UNIX (Beta under Windows 10)
  - UNIX (whether spelled Linux, AIX, HPUX, Solaris, etc., etc., etc.)
  - Cygwin under Windows
  - Proprietary (i.e., IBM's Z/OS)

Basic Problems

- What does the file look like at the beginning?
  - Need to sample a few rows
  - Need to perform special processing on first row (header)
- What does the file look at the end?
  - Need to see what it looks like at the end
  - Need to perform special processing on last row (trailer)
- Have a bad record too far in file to view in editor
- Get a file in another character set (like EBCDIC)
- Get a file that contains binary data
- Get a file that breaks your tool (XML Engine, ETL Tool)
- Need to understand the layout
- Need to perform basic sanity checks

- Do you really trust your data vendor?

What does the file look like?

- You need to see a few screens worth
- You don't want to fire up an editor
  - And wait for it to load the entire huge file
  - And you really don't want to write any code

```
more FILENAME

less FILENAME

pg FILENAME
```

- Will display the file a screen full at a time
- Which command is available varies by your distribution

What does the file look like at the beginning?

- You just need to see the first record (or first few)
- You don't want to fire up an editor
  - And wait for it to load the entire huge file
  - And you really don't want to write any code

```
head FILENAME

head -2 FILENAME

head -n 2 FILENAME
```

- Prints

```
LINE 1


LINE 1
LINE 2
```

What does the file look like at the end?

- You just need to see the last record (or last few)
- You don't want to fire up an editor
  - And wait for it to load the entire huge file
  - And you really don't want to write any code

```
tail FILENAME

tail -2 FILENAME

tail -n 2 FILENAME
```

- Prints

```
LAST LINE
```

```
LINE 99
LAST LINE
```

Have a bad record too far in file to view in editor

- You just need to see a specific record (or few records)
- You don't want to fire up an editor
  - And wait for it to load the entire huge file
  - And you really don't want to write any code
- I want records near #50 or just #50

```
head -51 FILENAME | tail -3

head -50 FILENAME | tail -1
```

- Prints

```
LINE 49
LINE 50
LINE 51


LINE 50
```

Have a bad record too far in file to view in editor

- I don't know the bad record but I know something about it
- Best when that something is a unique key

- Much like old DOS/CMD find command:

```
grep -i "line 50" FILENAME

grep  "LINE 50" FILENAME
```

- Prints

```
LINE 50


LINE 50
```

Saving Results

- Remember redirection and pipes in DOS/CMD?
- Well, they came from UNIX!
  - > sends output ("STDOUT") to a new file
  - >> appends output ("STDOUT") to an existing file, creates new if missing
  - 2> and 2>> sends error messages ("STDERR") to a file
  - | sends output ("STDOUT") to the input of another command
  - < gets "keyboard" input ("STDIN") from a file
  - << gets "keyboard" input ("STDIN") from the current script

Get a file in another character set (like EBCDIC)

- When I look at the file, I see garbage:

```
▒▒▒▒▒@▒%▒▒▒▒▒@▒%▒▒▒▒▒@▒%▒▒▒▒▒@▒%▒▒▒▒▒@▒%▒▒▒▒▒@
```

  - I can look at it in hex:

```
od -c -x ebFILE
0000000   ▒   ▒   ▒   ▒   @   ▒   %   ▒   ▒   ▒   ▒   @   ▒   %   ▒   ▒
              d3c9    d5c5    40f1    25d3    c9d5    c540    f225    d3c9
etc.
```

  - I can convert the entire file to ASCII:

```
dd if=ebFILE conv=ascii
LINE 1
LINE 2
LINE 3
etc.
```

- Imperfect solution if there is binary, zoned decimal, or packed decimal in the file

Get a file that contains fixed record length

- I often have to work with "blocked" or "fixed record length" files.
- Does not contain record separators. While SAS® can work with this data, it is hard to use other tools to understand the data
- Looks like:

```
LINE 01LINE 02LINE 03LINE 04LINE 05LINE 06LINE 07
```

  - But we know each data record is 7 bytes long

- Need to convert it to "variable length" with separators

```
dd conv=unblock cbs=7 if=INFILE
```

  - Gives:

```
LINE 01
LINE 02
LINE 03
LINE 04
LINE 05
LINE 06
LINE 07
```

Get a file that contains fixed record length

- I have a few options to make my life easier
  - Especially if I want to work with a subset of the file

```
count=n number of blocks to copy
ibs=n   bytes per record
skip=n  number of blocks to skip first
cbs=n   unblock bytes per record
```

- You can specify the filenames:

```
if=INPUT_FILE
of=OUTPUT_FILE
```

- Can be combined with ASCII conversion

```
dd conv=unblock,ascii
```

  - Just remember what I said about binary data in your file...

Get a file that contains binary data

- Files that contain binary data aren't exactly "human readable"

- You can write code to load, but sometimes you need to look at it.

- Object Dump (`od`) comes in handy

- When dealing with binary data in EBCDIC
  - One approach is a hybrid
    - Use `dd` perform the conversion on the entire file
    - Read both the original file and the converted file
    - Use the original file for binary, packed, zoned decimal fields
    - Use the converted file for text fields

  - The other is to perform the conversion in your own program

Get a file that breaks your tool (XML Engine)

- Our vendor included a tag that broke our XML Engine:

```
<?xml version="1.0"?>
```

- How to fix?
  - Vendor thought file was fine
  - XML Engine worked fine without the tag
  - So I removed it from the input file:

```
awk '{gsub("\\<\\?xml version=\"1\.0\"\\?>", "");print $0;}'
FNAME > XMLNAME
```

  - Simply put:
    - convert every appearance (global substitute) in a record of that string to nothing and output the resulting record.
    - In `awk`, $0 is the entire record as a string and there is an implied read loop.
    - Input is FNAME, output is XMLNAME.

  - To process a 4,720,941,011 byte file takes about 4 minutes using about 1/3 CPU core – 19 MB/second

Get a file that breaks your tool (ETL Tool)

- Our ETL tool was faulting on random records on NUL characters
  - Feed from our transaction processing system
  - Which got it from the Web interface

- How to fix?
  - Both teams disclaimed responsibility
  - No errors in their processing
  - ETL Vendor purposely included that error
  - So I removed it from the input file:

```
sed 's/\x0//g' < INFILE > OUTFILE
```

  - To process a 62,021,760 byte file takes about 10 seconds (CPU not noted) – 31 MB/second

- Could have used the translate command instead:

```
tr '\000' '' < INFILE > OUTFILE
tr -d '\000' < INFILE > OUTFILE
```

Get a file that breaks your tool (awk vs sed vs tr)

- Why did I use `awk` in the first example and `sed` in the second?
  - Both are standard tools
  - `sed` has a limited record length (around 4,000 bytes)
  - `awk` has a record length limit matching the C long size (or total available real/virtual memory).
  - `sed` uses less memory and is a bit faster
  - `sed` has a limited "programming" language – regular expressions
  - `awk` has a full feature programming language – C with Strings
  - I honestly forget why I didn't use `tr`
  - `sed ":w" < INFILE > OUTFILE`
  - Sometimes writing code is fun

- Remember when I asked "How about those with record lengths of almost 88,000 characters?"
  - That was the average record in my XML file
  - Oh, and my ETL tool limited normal text strings to 32,767 bytes

Get a file that breaks your tool (infile dsd dlm=",")

- New file received from vendor on UNIX server produced no errors in proc import but returned no records.
- The file did have contents – viewing with text editor or excel it looked fine:

```
key,val1,val2
123,1,2
123,3,4
125,5,6
```

- Using Object Dump ('`od`') showed:

```
0000000   k   e   y   ,   v   a   l   1   ,   v   a   l   2  \n   1   2
           656b    2c79    6176    316c    762c    6c61    0a32    3231
0000020   3   ,   1   ,   2  \n   1   2   3   ,   3   ,   4  \n   1   2
           2c33    2c31    0a32    3231    2c33    2c33    0a34    3231
0000040   5   ,   5   ,   6  \n
           2c35    2c35    0a36
```

- However, termstr=crlf expected:

```
0000000   k   e   y   ,   v   a   l   1   ,   v   a   l   2  \r  \n   1
           656b    2c79    6176    316c    762c    6c61    0d32    310a
0000020   2   3   ,   1   ,   2  \r  \n   1   2   3   ,   3   ,   4  \r
           3332    312c    322c    0a0d    3231    2c33    2c33    0d34
0000040  \n   1   2   5   ,   5   ,   6  \r  \n
           310a    3532    352c    362c    0a0d
```

Need to understand the layout

- There are a few ways to look at a file to determine the layout
- Object Dump (‘`od`’)
  - -x – hexadecimal
  - -o – octal
  - -a – Characters with named control characters
  - -c – Characters with escaped control characters
  - Can search output for specific control characters
- `strings` is one command I'll use even on binary (and executable files):
  - Displays any collection of 3 or more printable characters
- A little more complex if I need to look at specific fields

Need to understand the layout: Pulling Fields

- The vendor tells me the layout looks like:

LATITUDE	2563	2570	8	N	The property location based upon the "latitude" component of latitude/longitude coordinates. The latitude is stored in decimal degrees to 6 decimal places (e.g., 12.123456) and will always be positive north of the North American continent.
LONGITUDE	2571	2579	9	N	The property location based upon the "longitude" component of latitude/longitude coordinates. The longitude is stored in decimal degrees to 6 decimal places (e.g., 123.123456) and will always be negative on the North American continent.

- But my SAS® reports errors on those fields while reading as numeric

- I have a few choices:

```
cut -b 2563-2570,2571-2579 FILENAME
awk '{print " lat " substr($0, 2563, 8), " long " substr($0,
2571, 9);}' FILENAME
```

- Results in (respectively)

```
4271275707387955R          lat 42712757  long 07387955R
4271341007387929K          lat 42712757  long 07387955R
4271020507387605N          lat 42710205  long 07387605N
```

Need to understand the layout: Pulling Fields

- Results in (respectively)

```
4271275707387955R          lat 42712757  long 07387955R
4271341007387929K          lat 42712757  long 07387955R
4271020507387605N          lat 42710205  long 07387605N
```

- Turns out that, contrary to the document:
  - Latitude is COBOL zoned decimal: 99v9(6), USAGE DISPLAY
  - Longitude is COBOL signed zoned decimal: S999v9(6), USAGE DISPLAY

Sources of More Information

- Built in Manual:

```
man man
man -k transfer
```

  - Check the manual for a specific command or keyword search
  - May not be as good as a web search but provides details for ***that*** version
  - I emphasize its usage to my students
  - There's a reason everyone in IT knows what "RTFM" means!

- Most commands will also provide help if you provide an invalid option, use -?, or use - -help as parameters

- Of course, there's always Google or Bing...

Sources of More Information: Command Line Help

- For instance, with od

```
od -?
```

  - Resulting output (GNU):

```
od: unknown option -- ?
Try `od --help' for more information.
```

  - Or (Solaris)

```
usage: od [-bcCdDfFoOsSvxX] [-] [file] [offset_string]
       od [-bcCdDfFoOsSvxX] [-t type_string]... [-A
address_base] [-j skip] [-N count] [-] [file...]
```

Sources of More Information: Command Line Help

- For instance, with od (GNU)

```
od --help
```

  - Resulting output (just sample):

```
Usage: od [OPTION]... [FILE]...
  or:  od [-abcdfilosx]... [FILE] [[+]OFFSET[.][b]]
  or:  od --traditional [OPTION]... [FILE] [[+]OFFSET[.][b]
[+][LABEL][.][b]]

Write an unambiguous representation, octal bytes by default,
of FILE to standard output.  With more than one FILE argument,
concatenate them in the listed order to form the input.
With no FILE, or when FILE is -, read standard input.
```

Wrap Up

?! ! ! ?!

Questions

? ?

and

? ?

Answers

?! ! ! ?!

We still have some more time?

- We can talk about installing Windows 10 Bash: 20161022 Leveraging unix tools-Win_sub_Linux_beta.pptx

- We can look at Cygwin, Windows 10 Bash, and Linux under VirtualBox

- Or we can look at code:
  - How to perform basic sanity checks
  - A trick to make my life easier
  - Some UNIX Basics
    - Directory Navigation
    - Searching Data
    - Directory Contents
    - Other Commands

Need to perform basic sanity checks

- Unfortunately, data providers can't be trusted
  - Vendor
  - Internal

- So you really need to protect yourself with Sanity Checks
  - Header/trailer processing
  - Raw Record counts
  - Key Record counts
  - Record lengths
  - Field counts
  - Breaking up a file

Need to perform basic sanity checks: Header/Trailer

- Header/trailer processing
  - Can be fairly complex with both counts and sums
  - General process:
    - Read header record, save important values
    - Count data records and sum appropriate fields
    - Read trailer record, compare important values with your calculated
    - Abort on differences
  - May be better in your favorite programming tool
  - Sometimes all you have is a hammer
    - ACH inbound/outbound file validator: 1300+ lines of `awk` code
      - File header/trailer – Credit & Debit sums, hash, record counts
      - Batch header/trailer – Credit & Debit sums, hash, record counts
      - Various parent/child record combinations (limited to 94 byte records)
      - And check digits on various records/fields
    - You can always convert to C (awk2c) or Perl (awk2perl)!

Need to perform basic sanity checks: Raw Record Counts

- Need to count number of records in the file
- Can let your application perform the work or if checking manually:

```
wc -l FILE


wc FILE
```

- Resulting output (just sample):

```
100 FILE

100     200     793 FILE
```

Need to perform basic sanity checks: Key Record Counts

- Need to count number of records with same value in first field
- Would be a royal pain to find records manually

```
$ sort INPUT_FILE | awk
'BEGIN{c=0;o="XXXXXXXXXXXXXXXXXX";} {if ($1 != o) { print
c,o; o=$1; c=1; } else c++;} END{print c,o,"end";}' |
sort > OUTPUT_FILE
```

- Resulting output (just sample):

```
10 ONE
15 TWO
15 Three
16 Four
```

Need to perform basic sanity checks: Key Record Counts

- Let’s break that command line down
  - Sort the input (to put all values for same first field together)

```
sort INPUT_FILE |
```

- Execute awk to count records with same value (control break)

```
awk 'BEGIN{c=0;o="XXXXXXXXXXXXXXXXXX";} {if ($1 != o) {
print c,o; o=$1; c=1; } else c++;} END{print c,o,"end";}'
|
```

- Sort the output so they’re in count order

```
sort > OUTPUT_FILE
```

Need to perform basic sanity checks: Key Record Counts

- Let’s break the awk program down:

```
BEGIN{
   c=0;
   o="XXXXXXXXXXXXXXXXXX";}
{
   if ($1 != o)
   {
      print c,o;
      o=$1; c=1;
   }
   else c++;
}
END{print c,o,"end";}
```

Need to perform basic sanity checks: Record Lengths

- Need to find or catch specific “bad” records to prevent bad loads:

```
Source file contains: 67099, destination table contains:
66979
```

```
FR_3016 Row [346]: Record length [18398] is longer than line
sequential buffer length [3005] for FILENAME.dat.  Record
will be rejected.
```

- We bet there is more than one bad record
- Would be a royal pain to find records manually

```
awk '{ if (length($0) > 3004) print NR, length($0);}'
FILENAME.dat
```

- Resulting output (just sample):

```
346 18398
10000 10000
10101 12357
13031 8192
```

Need to perform basic sanity checks: Field Counts

- Need to validate number of fields on each record in a delimited file
  - Can choose the delimiter

- Would be a royal pain to check fields manually

```
awk 'BEGIN {FS=","} {print "NF " NF;};' csv1.csv

awk 'BEGIN {FS=","; max=0;} {print "NF " NF; if (NF >
max) max=NF;} END {print "max " max;}' csv1.csv
```

- Resulting output (just sample):

```
NF 9
NF 10
NF 9
NF 10
NF 9
max 10
```

Need to perform basic sanity checks: Breaking up a file

- Already reviewed field and byte position examples using `awk` and `cut`
- Another would be to split records to a more readable size
- Remember my 88,000 byte record length?
- Splitting those records by individual tags

```
awk 'BEGIN {FS="<";} {for (i=1;i<=NF;i++) if (length($i)
> 0) print "REC=" NR " Field=" i " : <" $i;}' FILENAME
```

- Or just the tags

```
awk 'BEGIN {FS="<";} {for (i=1;i<=NF;i++) if (length($i)
> 0) print "<" $i;}' FILENAME
```

- Resulting output (just sample):

```
REC=1 Field=12 : <wpt lat="40.224917" lon="-75.774154">
REC=1 Field=13 : <name>031201 CAR
REC=1 Field=14 : </name>
REC=1 Field=15 : <sym>Pin, Blue
```

Making my job easier

- Need to create set of commands to execute on a series of files (history load)
- Each execution needs to use the prior and current day files
- Would be a royal pain to build commands by hand

```
find . -name "ZCFPAYEE_FILE_[0-9][0-9][0-9][0-9][0-9].dat" -print | sort | awk 'BEGIN {last="";} {if (last == "") last=$1; else print "parse_diff.ksh " last " " $1; last=$1;}'
```

- Resulting output (just sample, not first or last ones):

```
parse_diff.ksh ./ZCFPAYEE_FILE_62285.dat ./ZCFPAYEE_FILE_62286.dat
parse_diff.ksh ./ZCFPAYEE_FILE_62286.dat ./ZCFPAYEE_FILE_62287.dat
parse_diff.ksh ./ZCFPAYEE_FILE_62287.dat ./ZCFPAYEE_FILE_62288.dat
parse_diff.ksh ./ZCFPAYEE_FILE_62288.dat ./ZCFPAYEE_FILE_62289.dat
```

Making my job easier

- Let’s break that command line down
  - Locate the files we want (only production files) and pipe out the name

```
find . -name "ZCFPAYEE_FILE_[0-9][0-9][0-9][0-9][0-
9].dat" -print |
```

  - Sort the result so they are in order (just in case)

```
sort |
```

  - Execute awk with a small program to build the command line

```
awk 'BEGIN {last="";} {if (last == "") last=$1; else
print "parse_diff.ksh " last " " $1; last=$1;}'
```

Making my job easier

- Let’s break the awk program down:

```
BEGIN {
    last="";
}

{
    if (last == "")
        last=$1;
    else
        print "parse_diff.ksh " last " " $1;
    last=$1;
}
```

Some UNIX Basics: Directory Navigation

- You will find UNIX directory structure very familiar
  - Windows shares many similarities include tree-metaphor structure
  - Use / instead of Windows' \
  - No drive letters
    - All disks are attached to the main directory structure ("mounted")
    - DOS 3.1 – 6 included the 'join' command that would do the same
    - Cygwin and Windows 10 Bash are special cases

- Directory Navigation
  - '`cd`' – change directory
  - '`cd`' with no parameters – change to home directory
  - '`cd -`' – change to previous directory (where you were before)
  - '`pwd`' – display the current directory (Windows '`cd`' does this)

Some UNIX Basics: Searching Data

- ‘`grep`’ – this is the command when you want to search the contents of a file
  - General form is ‘grep OPTIONS “search string” file_name_wildcarded’
  - Common OPTIONS
    - -v (invert match – show lines that don’t match)
    - -i (case insensitive search)
    - -H (show file names)
    - -n (show line number)
    - -r (recurse through subdirectories)
  - “search string” includes regular expressions
    - Entire books have been written to explain regular expressions!
    - ^ (beginning of line)
    - $ (end of line)
    - . (any single character)
    - [ABCabc] (single character that matches any of A, B, C, a, b, c)

Some UNIX Basics: Directory Contents

- ‘`ls`’ is the primary command for looking at files
  - Many options are available
  - -a – show all files; by default, files that begin with “.” are “hidden”
  - -l – long display showing full information
  - -1 – list in one column
  - -t – sort by time
  - -r – reversed (used with -t)
  - -R – recurse through subdirectories
  - Watch for “`Arg list too long`” – your wildcard list got too large
    - Windows requires individual programs to expand wildcards
    - UNIX shell expands wildcards “on the command line”

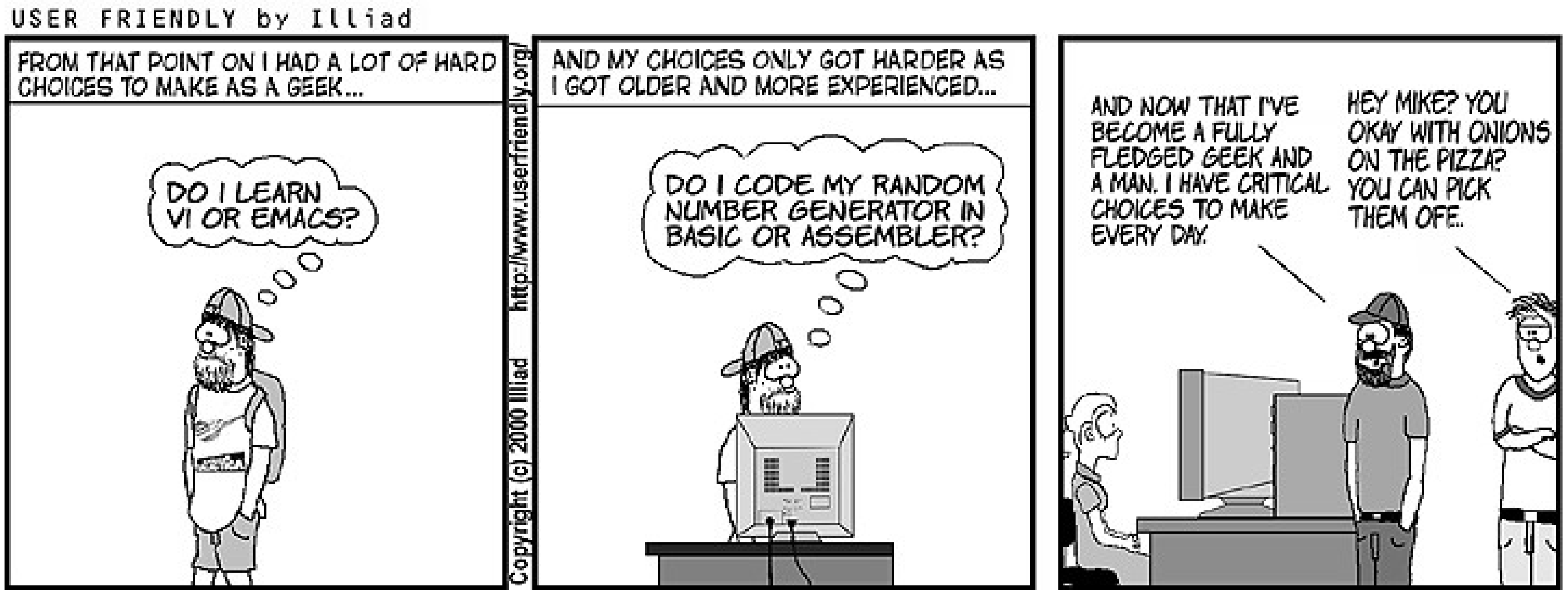
Some UNIX Basics: Other Commands

- Some other useful commands
  - ‘`cp`’, ‘`mv`’, ‘`rm`’ – copy, move or rename, or delete (remove) a file
  - ‘`mkdir`’, ‘`rmdir`’ – create (make) or delete (remove) a directory
  - ‘`exit`’, ‘`logout`’ – leave the system
  - ‘`du`’, ‘`df`’ – display storage usage or filesystem available

- Regular Expressions to match file names and data

Some UNIX Basics

- It is all about choices


USER FRIENDLY by Illiad
FROM THAT POINT ON I HAD A LOT OF HARD CHOICES TO MAKE AS A GEEK...
DO I LEARN VI OR EMACS?
http://www.userfriendly.org/
Copyright (c) 2000 Illiad
AND MY CHOICES ONLY GOT HARDER AS I GOT OLDER AND MORE EXPERIENCED...
DO I CODE MY RANDOM NUMBER GENERATOR IN BASIC OR ASSEMBLER?
AND NOW THAT I'VE BECOME A FULLY FLEDGED GEEK AND A MAN, I HAVE CRITICAL CHOICES TO MAKE EVERY DAY.
HEY MIKE? YOU OKAY WITH ONIONS ON THE PIZZA? YOU CAN PICK THEM OFF.

Wrap Up for Real

?! ! ! ?!

? **Questions** ?

and

? ?

?! **Answers** ! ! ?!