CS571 Advanced Programming Techniques

Awk

Make

Today's agenda

- Discuss solution points to Assignment 4
- Text Manipulations (vi, sed)
- vi and what to do about TABs
- AWK
- make

Discussion points on Assignment 4

- You can access the arguments to a script using the special shell variables
  - `$1, $2, … $#`
  - which you can override using the set command
  - `set arg1 arg2 arg3`
  - you cannot override $0

- The test below checks to see if the string `database` is empty

```
if test "X$database" = "X"
```

- To read a file and output only the lines that satisfy a condition (regexp), use

```
cat $database | egrep condition
```

Search and replace using `vi`

```
:%s/search/replacement/g
```

`:` go to the command line

`%` to substitute all lines, could also use

`5,10` substitute in lines 5-10

`.` substitute in current line only (same if . Is missing)

`+10` (substitute the next 10 lines)

`search` a regular expression that describes the string to change

`replacement` what we replace the string with

`g` make the substitution globally in the qualifying lines

default changes only the first occurrence

Search and Replace using `sed`

- `sed` stands for stream editor and it can be used for text manipulations (searching, search and replacing, inserting and deleting)
- Mostly used for searching and replacing, e.g.,

```
sed 's/word1/word2/g' input.file > output.file
```

- replaces every instance of `word1` with `word2` in `input.file` and stores the output in `output.file`

vi and TAB (add these to .vimrc or .exrc)

```
:set expandtab (noexpandtab)
```

all the tabs entered after this command is executed will be replaced by spaces

```
:retab
```

change the existing tab characters to match the current tab settings

```
:set tabstop = 4
```

all tab characters are 4 spaces

vi and TAB (add these to .exrc)

- For python programming

```
:set expandtab
:set tabstop=4
```

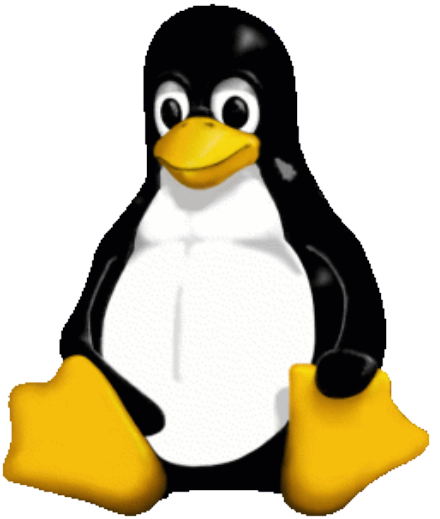
- For make files

```
:set noexpandtab
```

AWK, 1977

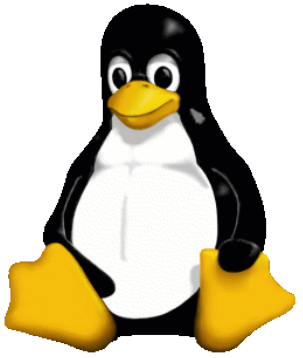
- Alfred **A**ho, Peter **W**einberger, Brian **K**ernighan
- Simple, mechanical data manipulations
  - Change format
  - Check data validity
  - Find items with some properties
  - Add numbers
  - Print reports
- Very short programs (one liners!)
- AWK – program

What's my name?



Tux – The Linux penguin

- Tux is a penguin character
- The official brand character of the Linux kernel
- Originally created as an entry to a Linux logo competition

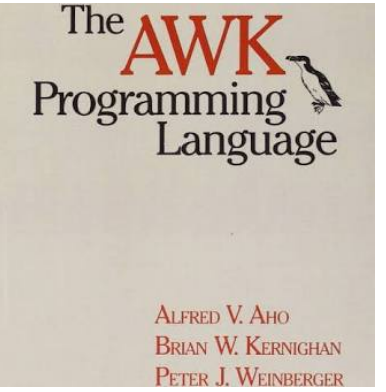


Tux

(T)orvalds (U)ni(X)"

1996

Linus Torvalds Favorite Picture



1988 AWK book

1988 AWK book

AWK

- AWK searches an input file for lines that match a pattern
  - Regular expressions
  - Comparisons on strings, numbers, arrays, variables, etc.
- For every matching line, a corresponding action is performed
- AWK splits the input line into fields automatically

Getting started

- Supposed we have the `emp.data` file (name, hours, payrate)

```
Beth    4.00    0
Dan     3.75    0
Kathy   4.00    10
Mark    5.00    20
Mary    5.50    22
Susie   4.25    18
```

$1 name
$2 hours
$3 payrate

- What does this program do?

```
awk '$3 > 0 { print $1, $2 * $3 }' emp.data
```

Getting started

- Supposed we have the `emp.data` file (name, hours, pay)

```
Beth    4.00    0
Dan     3.75    0
Kathy   4.00    10
Mark    5.00    20
Mary    5.50    22
Susie   4.25    18
```

pattern to match

action to do

- What does this program do?

```
awk '$3 > 0 { print $1, $2 * $3 }' emp.data
```

- `$1` is the first field of each line, `$2` the second, `$3` the third
- It outputs the earnings for employees who worked

```
Kathy   40
Mark    100
Mary    121
Susie   76.5
```

Getting started

- Supposed we have the `emp.data` file (name, hours, pay)

```
Beth    4.00    0
Dan     3.75    0
Kathy   4.00    10
Mark    5.00    20
Mary    5.50    22
Susie   4.25    18
```

- What does this do?

```
awk '$3 == 0 { print $1 }' emp.data
```

Getting started

- Supposed we have the `emp.data` file (name, hours, pay)

```
Beth    4.00    0
Dan     3.75    0
Kathy   4.00    10
Mark    5.00    20
Mary    5.50    22
Susie   4.25    18
```

- What do this do?

```
awk '$3 == 0 { print $1 }' emp.data
```

- Lists names of people who did not work

```
Beth
Dan
```

`awk` programs

- Each `awk` program is a sequence of

```
pattern { actions }
```

- `awk` scans a sequence of input lines, searching for lines that are matched by any of the patterns in the program
- Every input line is checked against each pattern
- For each pattern that matches, the corresponding action(s) are performed

Form of an `awk` Program

- An `awk` program is a sequence of function definitions and one or more rules :

```
pattern {actions}
```

- `pattern` – Numeric or string relational operator, or regular expression match
  - If empty, actions are applied to every record
- `action` – Statement or sequence of statements
  - If empty, default action is to print the entire line

The pattern and/or action may be missing

All valid AWK programs

```
$3 == 0 { print $1 }
```

```
$3 == 0
```

```
{ print $1 }
```

How to run AWK

- From the command line

```
awk program inputFiles
awk '$3 == 0 { print $1 }' file1 file2
```

- From the command line without an input file

```
awk 'program'
```

- Using a file as input

```
awk -f programfile optional list of input files
```

- From a bash file

```
!/bin/bash
awk '$3 == 0 { print $1 }' file1
```

Note the single quotes

What do these programs do?

```
{ print }
```

What do these programs do?

`{ print }` print every line

```
{ print  $0 }
```

What do these programs do?

`{ print }` print every line

`{ print $0 }` also print every line

```
{ print $1, $2 }
```

What do these programs do?

`{ print }` print every line

`{ print $0 }` also print every line

`{ print $1, $2 }` print certain fields

`{ print NF, $1, $NF }`

What do these programs do?

`{ print }` print every line

`{ print $0 }` also print every line

`{ print $1, $2 }` print certain fields

`{ print NF, $1, $NF }` NF=number of fields, $NF = last field

```
{ print $1, $2 * $3 }
```

What do these programs do?

`{ print }` print every line

`{ print $0 }` also print every line

`{ print $1, $2 }` print certain fields

`{ print NF, $1, $NF }` NF=number of fields, $NF = last field

`{ print $1, $2 * $3 }` compute and print

```
{ print NR, $0 }
```

What do these programs do?

`{ print }` print every line

`{ print $0 }` also print every line

`{ print $1, $2 }` print certain fields

`{ print NF, $1, $NF }` NF=number of fields, $NF = last field

`{ print $1, $2 * $3 }` compute and print

`{ print NR, $0 }` printing line numbers (number of records), $0 is entire line

```
{ printf("total pay for %s is $%.2f\n", $1, $2 * $3) }
```

What do these programs do?

`{ print }` print every line

`{ print $0 }` also print every line

`{ print $1, $2 }` print certain fields

`{ print NF, $1, $NF }` NF=number of fields, $NF = last field

`{ print $1, $2 * $3 }` compute and print

`{ print NR, $0 }` printing line numbers (number of records), $0 is entire line

`{ printf("total pay for %s is $%.2f\n", $1, $2 * $3) }` – print $1 as a string (%s) and the result of $2*$3 as a number with 2 digits after the decimal point

```
{ printf("%-8s $%6.2f\n", $1, $2 * $3) }
```

Selection

- By comparison

```
$2 >= 5
```

- By computation

```
$2 * $3 > 50
```

- By text content

```
$1 == "Susie"
```

- By combinations of patterns

```
$2 >= 4 || $3 >= 20
```

- By negation

```
! ( $2 < 4 &.&. $3 < 20)
```

What do these patterns do?

```
$3>0              # print all lines where field 3 is greater than 0
$1=="Ben"         # Find Ben's record
/[Zz]+czc/        # print all lines that contain a match for RE
$5~/[Ww]aldo/ # print record if Waldo is hiding in field 5
```

Data Validation

```
NF I= 3 { print $0, "number of fields is not equal to 3" }
$2 < 7.25 { print SO, "rate is below minimum wage" }
$2 > 100 { print $0, "rate exceeds $100 per hour" }
$3 < 0 { print $0, "negative hours worked" }
$3 > 60 { print $0, "too many hours worked" }
```

What's Next

- Midterm Online Exam - The exam will cover all material covered during the first five weeks of classes. The exam can be taken any time between 02/12/20 at 6 PM and 02/16/20 at 11:59 PM. Examination is time limited to 90 minutes from the start of the test and will auto-submit after 90 minutes.
- Assignment 5 due February 18 at 11:59pm
- Try the examples and do the exercises from the following tutorials

Link to AWK Tutorial - Basic

Link to AWK Tutorial - CSV Files

AWK One Liners