Unix and Linux

More Shell Script Programming

Permanently modify your PATH

Use vi to edit your `.bash_profile` file

Add the following lines:

```
PATH=$PATH:~/bin
export PATH
```

Save the file and exit vi. Then, to execute the new commands in your .bash_profile, type

```
source .bash_profile
```

In the future, the commands in your .bash_profile will be executed whenever you login to your account.

Integer Comparisons

If you want to perform integer (numeric) comparison operations, use the following:

-eq True if values are equal to each other.

-ne True if values are not equal to each other.

-gt True if first value is greater than second value.

-lt True if first value is less than second value.

-ge True if first value is greater than or equal to second value.

-le True if first value is less than or equal to second value.

Integer comparison examples

```
if [ "$a" -eq "$b" ]
if [ "$a" -ne "$b" ]
if [ "$a" -gt "$b" ]
if [ "$a" -ge "$b" ]
if [ "$a" -lt "$b" ]
if [ "$a" -le "$b" ]
```

File Test Comparisons

You can also perform test operations on files and directories to determine information about their status. These operations include:

-d True if “file” is a directory.

-f True if “file” is a regular file.

-r True if “file” exists and is readable by the user running the script.

-w True if “file” exists and is writable by the user running the script.

-x True if “file” exists and is executable by the user running the script.

File Test Comparison Examples

```
if [ -d data ]
if [ -f data ]
if [ -r data ]
if [ -w data ]
if [ -x data ]
```

Flow Control Statements

Other flow control statements in addition to if statements,:

The `while` loop—Executes a block of code repeatedly as long as the conditional statement is true.

The `until` loop—Executes a block of code repeatedly as long as the conditional statement is false. Essentially the opposite of a while loop.

The `case` statement—Similar to an if statement but provides an easier branching method for multiple situations.

The `for` loop—Executes a block of code for each item of a list of values.

The `while` loop

Use vi to create a file named `zip.sh` in your `bin` directory containing the following lines and then test it:

```
#! /bin/bash
# zip.sh
echo "Enter a five-digit ZIP code: "
read ZIP

while echo $ZIP | egrep -v "^[0-9]{5}$" > /dev/null 2>&1
do
   echo "You must enter a valid ZIP code – five digits
only!"
   echo "Enter a five-digit ZIP code: "
   read ZIP
done
echo "Thank you"
```

The grep command

“The most powerful and useful for a software developer is the grep command”.

This command is designed to act as a filter, only displaying the lines of data that match a pattern.

Find all ocurrences of “the” in the /etc/bashrc file

```
grep “the” /etc/bashrc
```

Find only the occurences as a separate word

```
grep –w “the” /etc/bashrc
```

Regular Expressions
“wildcards on steroids”

Display lines that contain “if”

```
grep “if” /etc/bashrc
```

Display lines that start with “if”

```
grep “^if” /etc/bashrc
```

How many subdirectories does the /etc directory have?

```
ls –l /etc | grep “^d” | wc –l
```

How many files?

```
ls –l /etc | grep “^-” | wc –l
```

Some Regex characters

* Matches zero or more of the previous character.

. Matches any single character.

[ ] Matches a single character from a subset of characters;

[abc] matches either an a, b, or c

[0-9] matches any digit from 0 through 9

\ Escapes the special meaning of a regular expression character;

* matches simply a * character.

{x} means find x number of occurrences

$ means search at the end of the string

`egrep -v "^[0-9]{5}$"`

egrep is an extended version of grep

The –v option means return the lines that **don't** match

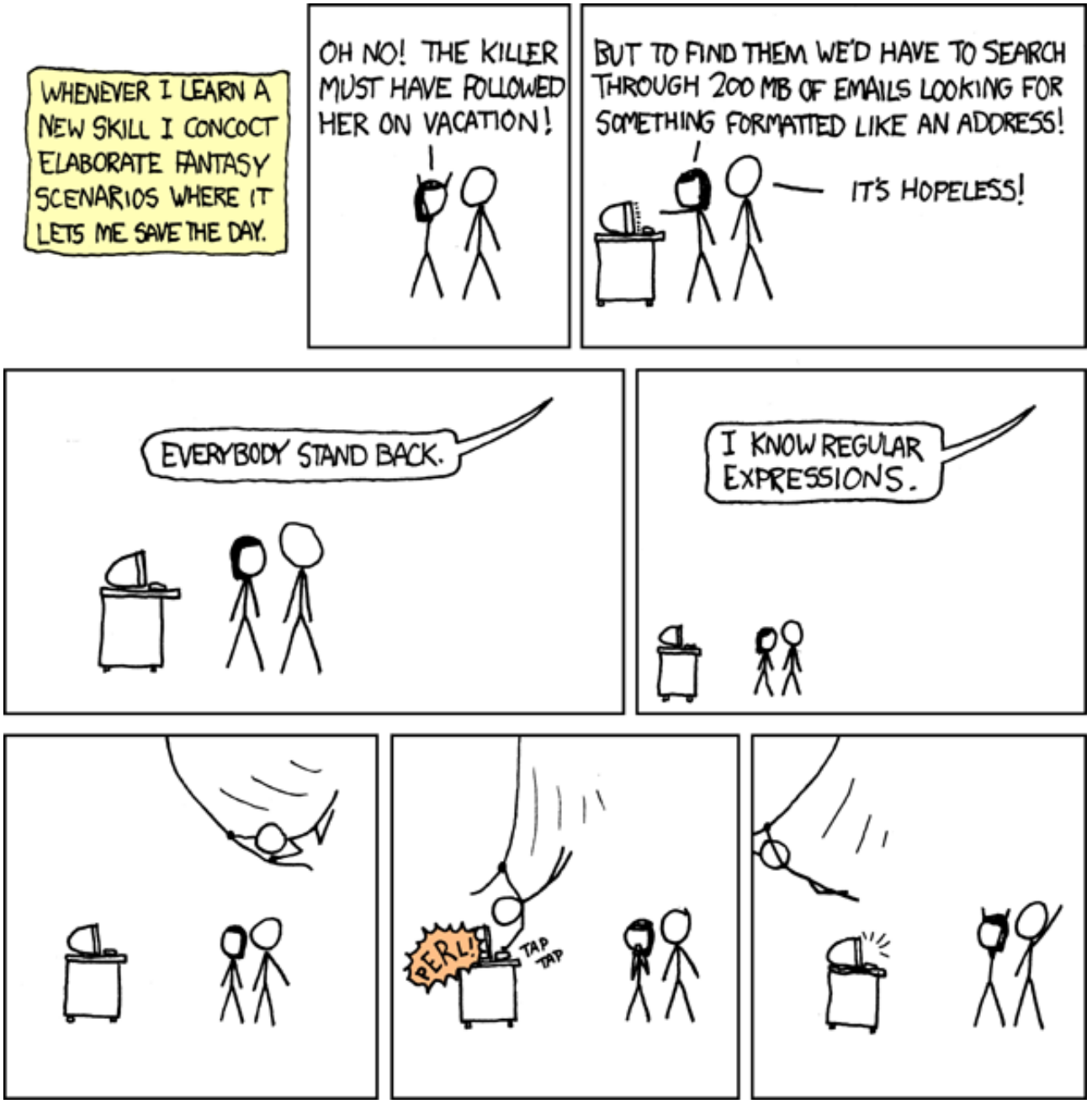
^ means the line must begin with the string

[0-9] the string must contain digits

{5} means the string must contain 5 characters

$ means the line must end with the string

`echo $ZIP | egrep -v "^[0-9]{5}$"` returns "true" if $ZIP consists of anything other than a 5 digit string


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.
PERL!
TAP TAP

> /dev/null 2>&1

Two parts:

We don’t want to see the output of the egrep command so

/dev/null redirects it’s output to “the trash”

2>&1 redirects errors to the standard output device (i.e. the screen)

for loop

The IFS environment variable is the internal field separator
Create the script `path.sh` with this code and test it:

```
#! /bin/bash
# path.sh
IFS=:

for dir in $PATH
do
      ls -ld $dir
done
```

Case syntax

```
case expression in
    pattern1 )
        statements ;;
    pattern2 )
        statements ;;
    ...
esac
```

`case` statement key points

- `case` statement first expands the expression and tries to match it against each pattern.
- When a match is found all of the associated statements until the double semicolon (;;) are executed.
- After the first match, case terminates with the exit status of the last command that was executed.
- If there is no match, exit status of case is zero.

`case` example

Create the script colors.sh and test it

```
#! /bin/bash
# menu
echo -n "Enter your favorite color: "; read color
 case "$color" in
    "blue") echo "As in My Blue Heaven.";;
    "yellow") echo "As in the Yellow Sunset.";;
    "red") echo "As in Red Rover, Red Rover.";;
    "orange") echo "As in Autumn has shades of Orange.";;
    *) echo "Sorry, I do not know that color.";;
 esac
```

Shell Variables

- Basic guidelines for handling shell variables:
  - Omit spaces when you assign a variable without using single/double quotation marks around value
  - To assign a variable that must contain spaces, enclose value in “” or ‘’
  - To reference a variable, use a $ in front of it or enclose it in { }
  - Use [ ] to refer to a specific value in an array
  - Export a shell variable to make the variable available to other scripts
  - To make a shell variable read-only: *readonly fname*

Shell Variables (continued)

- Sample guidelines for naming shell variables:
  - Avoid using dollar sign in variable names
  - Use descriptive names
  - Use capitalization appropriately and consistently
  - If a variable name is to consist of two or more words, use underscores between the words