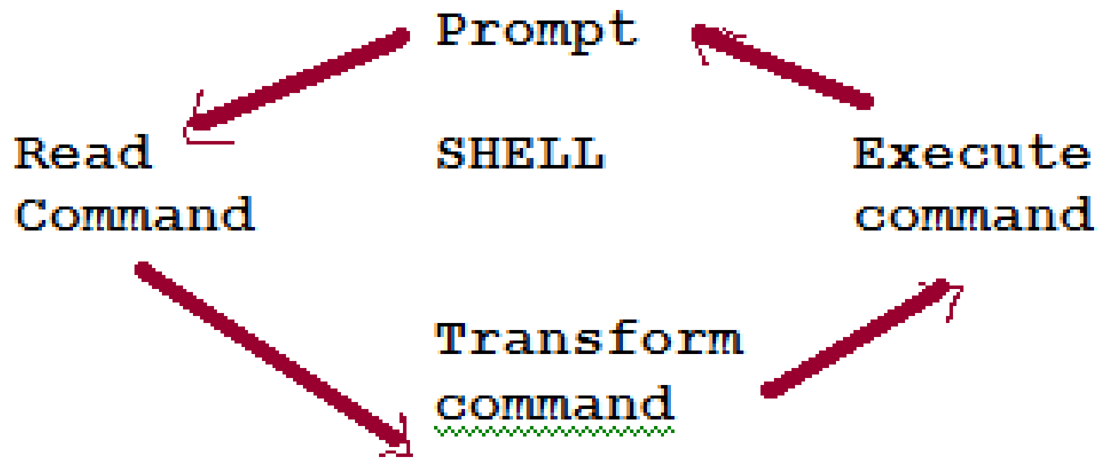
Shell Programming

15-123

Systems Skills in C and Unix

The Shell

- A command line interpreter that provides the interface to Unix OS.



Prompt

Read Command

SHELL

Execute command

Transform command

What Shell are we on?

- **echo $SHELL**
- Most unix systems have
  - Bourne shell (**sh**)
    - No command history
  - Korn shell (**ksh)**
    - Shell functions
  - C shell (**csh**)
    - History, no shell functions
- More details at unix.com

A Shell Script

```
#!/bin/sh
-- above line should always be the first line in your script
# A simple script
who am I
Date
```

- *Execute with: sh first.sh*

= ./first.sh

Things to do with shell scripts

- Remove all empty folders
- Remove all duplicate lines from a file
- Send email if the assignment is not submitted
- Check output of a submitted program against sample output
- Given a roster file, extract ID’s and create folders for each person
- Rename a folder that contains .txt files to a folder that contains all .htm files

Variables in shell

- System variables
  - $SHELL
  - $LOGNAME
  - $PWD
- User defined variables
  - name=guna — 'guna' — "guna"
  - echo “$name”

no spaces

Dir=$PWD/"guna"

mkdir $DIR

echo

- echo [options] [string, variables...]
- Options
  - -n Do not output the trailing new line.
  - -e enable interpretation
  - escaped special characters

    \a alert (bell)
    \b backspace
    \c suppress trailing new line
    \n new line
    \r carriage return
    \t horizontal tab
    \\ backslash

echo -e "hello \a"

Shell Variables

- echo $PATH – an environment variable
- **Environment variables can be changed**
  - **PATH=$PATH:*/usr/local/apache/bin*:.**
- **Examples**
  - **dir=~~pwd~~** $PWD
  - **echo $dir**
  - **subdir="lab1"**
  - **abspath=$dir/$subdir**

Command Line Arguments

echo "$1 $2"

- $# - represents the total number of arguments (much like argv) – except command
- • $0 - represents the name of the script, as invoked
- • $1, $2, $3, .., $8, $9 - The first 9 command line arguments
- • $* - all command line arguments OR
- • $@ - all command line arguments

Shell	perl
$#	$#ARGV+1

Using Quotes

- Shell scripting has three different styles of quoting -- each with a different meaning:
  - unquoted strings are normally interpreted

    guna

  - "quoted strings are basically literals -- but $variables are evaluated“

    "guna\n $name"

  - 'quoted strings are absolutely literally interpreted‘

    'guna\n $name'

  - `commands in quotes like this are executed, their output is then inserted as if it were assigned to a variable and then that variable was evaluated`

    `rmdir "$dirname"`

Examples

- ***day=`date | cut -d" " -f1`***
- ***printf "Today is %s.\n" $day***

Expressions

- **Evaluating Expr**
  - ***``sum=`expr $1 + $2` ``***
  - ***`printf "%s + %s = %s\n" $1 $2 $sum`***
- Special Variables
  - **$? - the exit status of the last program to exit**
  - **$$ - The shell's pid**
  - **Examples**
    - ***`test "$LOGNAME" = guna`***
    - ***`echo $?`***

expr

- Syntax: expr $var1 operator $var2
- Operators

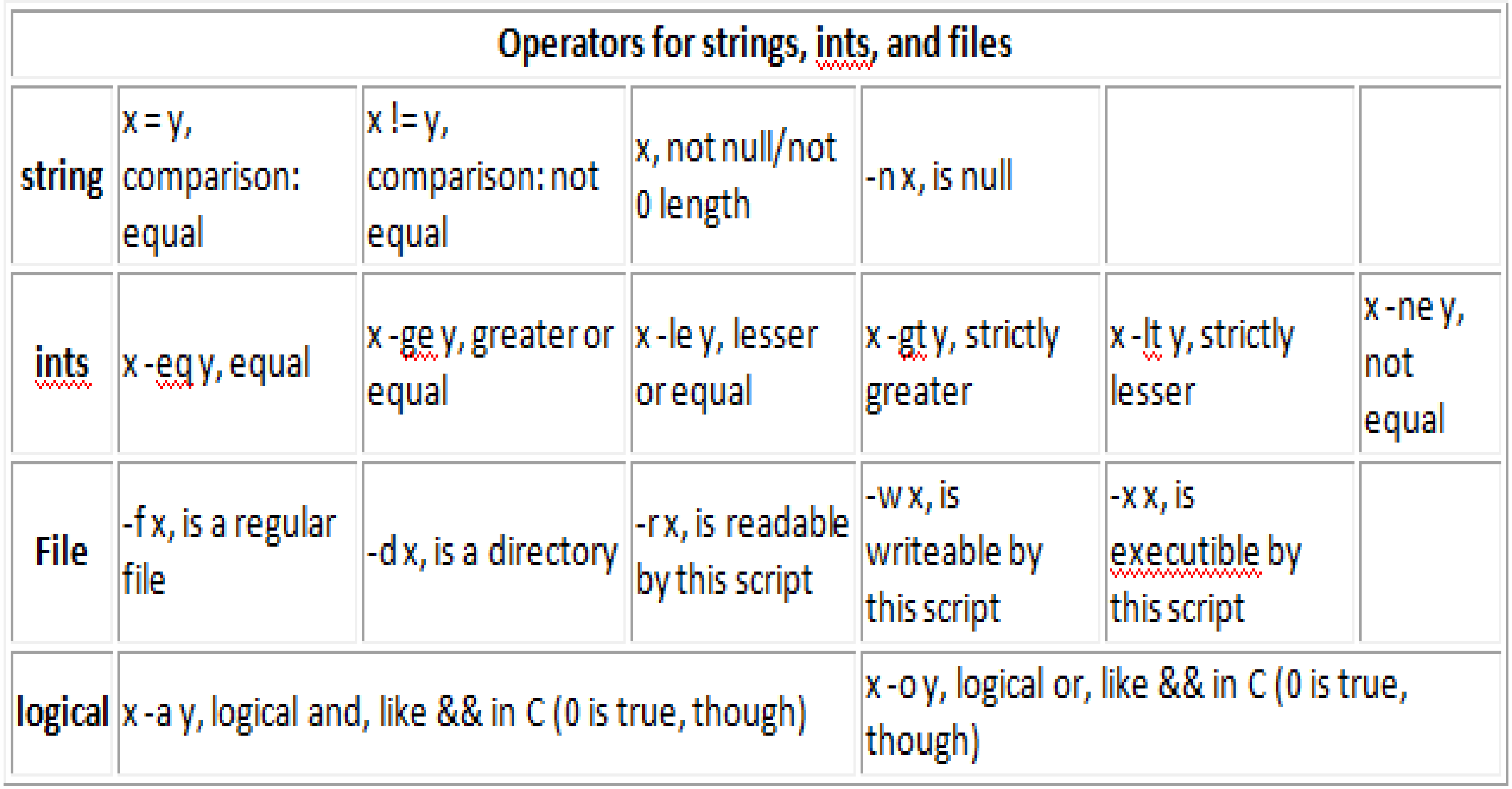
+ , - , / , % , *

wildcard

```
`expr $x % $y`

year=`date | cut -d" " -f5`
if [ `expr $year % 4` -eq 0 ]
  then
    ________
fi
```

Operators for strings, ints and files



Operators for strings, ints, and files						
string	x = y, comparison: equal	x != y, comparison: not equal	x, not null/not 0 length	-n x, is null		
ints	x -eq y, equal	x -ge y, greater or equal	x -le y, lesser or equal	x -gt y, strictly greater	x -lt y, strictly lesser	x -ne y, not equal
File	-f x, is a regular file	-d x, is a directory	-r x, is readable by this script	-w x, is writeable by this script	-x x, is executible by this script	
logical	x -a y, logical and, like && in C (0 is true, though)			x -o y, logical or, like && in C (0 is true, though)		

Conditionals

- **test -f somefile.txt**

  or

- **[-f somefile.txt]**

- **If statement**

Space Important

```
if [ "$LOGNAME"="guna" ]
 then
    printf "%s is logged in" $LOGNAME
else
   printf "Intruder! Intruder!"
fi
```

The for loop

```
for var in "$@"
   do
      printf "%s\n" $var
   done

for (( i = 1 ; i < 20 ; i++ ))
 do

done
```

Command line

```
sum=0
for var in "$@"
  do
    sum='expr $sum + $var'
done

> sh sum.sh 1 2 3
```

While loop

Process 1

Process 2

```
ls | sort |
 while read file
  do
     echo $file  | Sed 's/.txt/.htm'
  done
```

Command

I/O

- File descriptors
  - Stdin(0), stdout(1), stderror(2)
- Input from stdin
  - read data
  - echo $data
- redirecting
  - rm filename 1>&2

Functions

```
whologgedin()
{
   echo “hello $LOGNAME”
}
```

Calling:

```
> whologgedin
```

grep/sed/tr/s

- grep pattern file

grep Regex File

- sed s/regex1/regex2/
- sed tr/[a-z]/[A-Z]/

Calling shell commands from perl

- #! /usr/local/perl
- `mv $file1 $file2`;
- $out = `expr 1 + 2`;

Shell

Out=`expr 1 + 2`

Things to do with shell scripts

- Remove all empty folders
- Remove all duplicate lines from a file
- Send email if the assignment is not submitted
- Check output of a submitted program against sample output
- Given a roster file, extract ID’s and create folders for each person
- Rename a folder that contains .txt files to a folder that contains all .htm files

Coding Examples