# Introduction to Rexx

Prof. Chris GauthierDickey

COMP 2400: Unix Tools

UNIVERSITY OF
DENVER

# What is Rexx?

- Rexx is a programming language used primarily on the IBM mainframes, but also available on other platforms
  - e.g., regina on the Linux, OS X, and Windows platform
- It's a scripting language, to be fair, much like bash in functionality, but different syntactically

# Rexx

- Rexx is vital for z/VM as you can use it to build and glue CMS applications together
- Rexx is composed of:
  - operators, symbols, etc
  - A tiny core of instructions: only 20 or so
  - 70 built-in functions
  - the ability to execute external commands

UNIVERSITY OF DENVER

# Rexx on Linux

- Unlike bash, you don't have a Rexx shell on Linux
  - Instead, you have regina, a Rexx interpreter
- You have to write your code and execute it using regina
- Like all other scripts, you can begin your Rexx script with the #! to indicate the interpreter

# Rexx on Linux

- Your first Rexx program
- Note the #!/usr/bin/regina which specifies the interpreter
- /* */ denotes a comment
- say is a command like echo in bash

```
#!/usr/bin/regina
/* this is a rexx comment */
say "Hello World!"
```

UNIVERSITY OF DENVER

# Rexx Composition

- Rexx is made up of:
  - Instructions, which are keywords, assignments, labels, and commands
  - Built-in functions
  - System supplied functions

UNIVERSITY OF DENVER

# Variables

- A Rexx variable can consist of
  - [A-Za-z#$_][A-Za-z0-9#$_...])* <-- yes, a regex!
- RC, SIGL, RESULT are keywords you can't use
- You can't begin with a . or 0-9
- 250 chars is the max variable length

# Assignments

- We use = for assignment
  - x = 5
  - #9F3D = 'hello'
  - y = m * x + b
  - a = b

# Math

- Operators:
  - +, -, *, /: the usual
  - %: DIVIDE and drop the remainder
  - //: DIVIDE and only return the remainder
  - **: Exponentiate

```
#!/usr/bin/regina
say 5 + 6
say 10 % 3
say 10 // 3
say 10 ** 3
```

# Concatenation

- Putting a blank between values places a single blank between them in output
- Putting ll places no blanks between the items

```
#!/usr/bin/regina
w1='H'
w2='AL'
w3='is back'
/* note the multiple spaces,
   but it outputs only one
   space */
SAY w1||w2   w3
```

UNIVERSITY OF DENVER

# Comparison

- == is strictly equal
- = is equal
- \== is not strictly equal
- \= is not equal
- > greater than
- < less than
- >< greater than or less than

# Boolean Operators

- & returns 1 if they are both true, 0 otherwise
- | returns 1 if at least one is true, 0 otherwise
- && returns 1 if only one comparison (but not both) is true, 0 otherwise
- prefix \ returns the opposite response
  - \(5 = 4)

# IF-THEN-ELSE

- The if-then-else clause is just like you expect it to work
- It's good programming practice to use a NOP command at an ELSE that doesn't have a body
- Your if-expression must result in 1 or 0

```
#!/usr/bin/regina

PARSE ARG v1 v2
if v1 = 'hello'
  then say 'Goodbye'

if v2 = 'world' then
  do
    say 'universe!'
  end
else
  do
    say 'huh?'
  end
```

# SELECT

- Select is slightly different than most languages

```
#!/usr/bin/regina
SELECT
  WHEN v1 = 1 THEN say 'Got 1'
  WHEN v1 = 2 THEN say 'Got 2'
  WHEN v1 = 3 THEN say 'Got 3'
  OTHERWISE
    say 'Many'
END
```

# The DO loop

- The DO loop is like a for loop
- We can also loop 'forever'

```
#!/usr/bin/regina

parse arg x
DO i = 1 to x by 1
  say i
END

do FOREVER
  say 'Oh no!'
end
```

UNIVERSITY OF DENVER

# Exiting loops

- We can exit a loop with LEAVE, EXIT, or ITERATE
  - LEAVE terminates the loop and continues running
  - EXIT exits the script
  - ITERATE jumps back to the top of the loop, including reading the condition

# WHILE and UNTIL

- DO WHILE expression
  - lets us test an expression, which if true will continue to execute the loop
- DO UNTIL expression
  - lets us test an expression, which if false, will continue to execute the loop
  - UNTIL will NOT test until the END of the loop