# Shell scripting and system variables

HORT 59000

Lecture 5

Instructor: Kranthi Varala

# Text editors

- Programs built to assist creation and manipulation of text files, typically scripts.
- nano : easy-to-learn, supports syntax highlighting, lacks GUI.
- Emacs : provides basic editing functions but also extendible to add functionality. Supports GUI, extensions provide a wide range of functions.
- vi/vim : extensive editing functions and relatively limited extensibility, command and insert modes distinct, steep learning curve, but very rewarding experience.

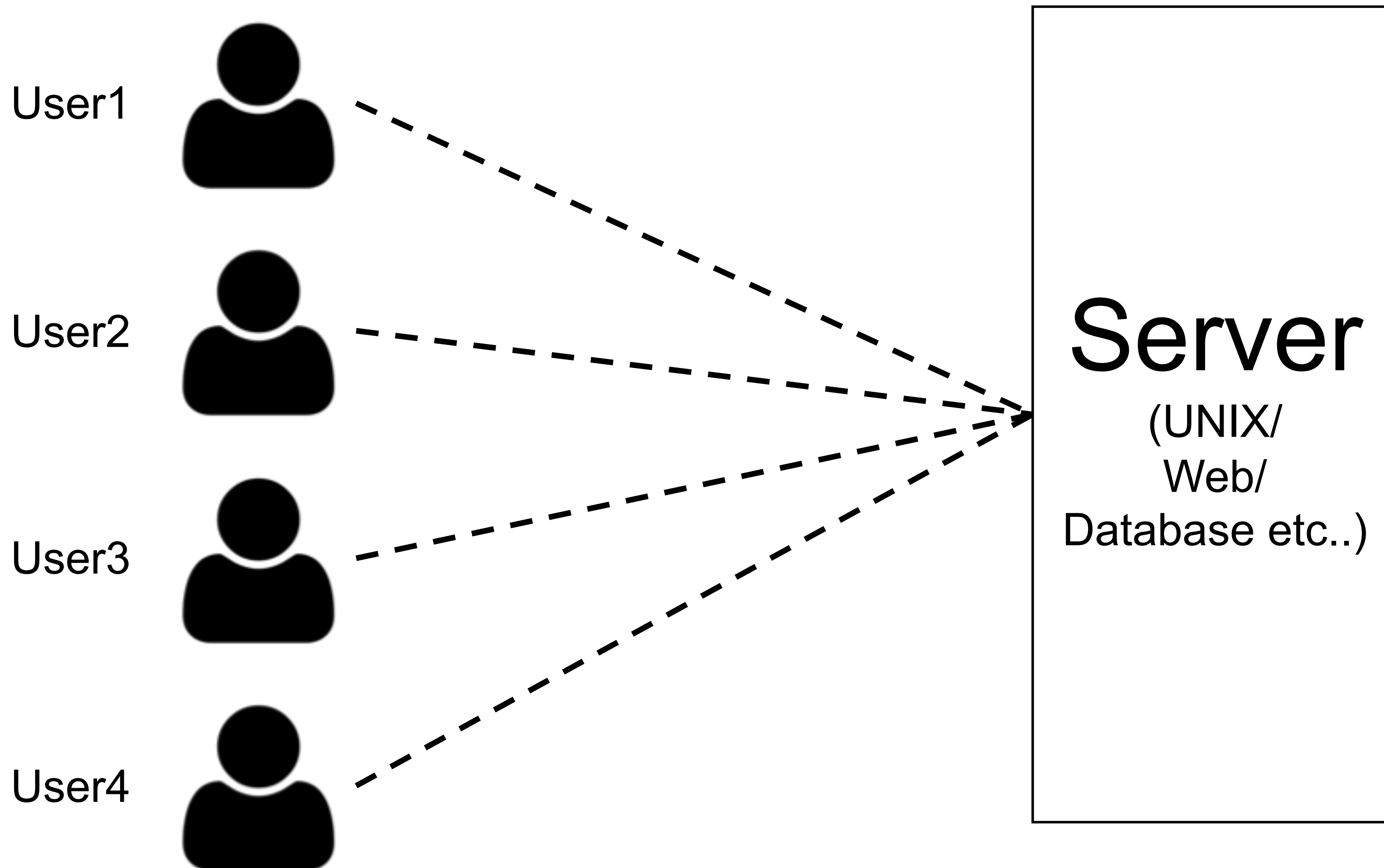
# Text manipulations

- Tabular data files can be manipulated at a column-level. 1. Cut: Divide file & extract columns. 2. Paste: Combine multiple columns into a single table/file.
- Sort: Sort lines in a file based on contents of one or more columns.
- Regular expressions : defining patterns in text. Special characters and quantifiers allow search and replacement of simple-to-complex matches.
- grep and awk use the power of regular expressions to make text processing very easy.

# Command-line operations

- All commands so far are run one at a time.
- Redirection and pipes allow combining a few commands together into a single pipeline.
- Lacks logical complexity, such as ability to make decisions based on input / values in file.
- Certain repetitive tasks are tedious to user.
- All commands are being sent to and interpreted by the ‘shell’

# Client/Server architecture



# Terminology

- Terminal: Device or Program used to establish a connection to the UNIX server
- Shell: Program that runs on the server and interprets the commands from the terminal.
- Command line: The text-interface you use to interact with the shell.

# Shells

- Shell itself is a program on the server and can be one of many varieties
  1. bash : Most popular shell, default on most Linux systems. Installed on all Linux systems
  2. zsh : A bash-like shell with some extra features. E.g., support for decimals, spelling correction etc.
  3. tcsh : A C-like syntax for scripting, supports arguments for aliases etc.
- We will work with bash shell scripting since it is the most common and supported shell.

# Environment variables

- A variable is a container that has a defined value.
- It's called a variable because the value contained inside it can change.
- Variables allow changing a part of the command that is to be executed.
- Every shell has a set of attached variables. See them by using the command `env`
- E.g., the variable `SHELL` contains the path to the current shell.

# Working with environment variables

- Set the value of a variable as follows:

```
FOO=BAR
```

- Retrieve the value of a variable as follows:

```
echo $FOO
```

# Example Environment variables

- On scholar: using the command `env` shows 99 environment variables:
- Examples:

```
HOME=/home/kvarala

SHELL=/bin/bash

HOSTNAME=scholar-fe01.rcac.purdue.edu

HISTSIZE=1000

RCAC_SCRATCH=/scratch/scholar/k/kvarala
```

# Environment vs. Shell variables

- Environment variables are ‘global’ i.e., shared by all shells started AFTER variable is defined.
- Shell variables are only present in the shell in which they were defined.
- Environment variables are inherited by child shells but shell variables are not.
- Shell variable can be made an environment variable by using `export` command.

```
FOO=BAR

export FOO
```

# Environment vs. Shell variables

| | |
|---|---|
| `$ export FOO=BAR` | (FOO defined in the environment) |
| `$ FOO2=BAR2` | (FOO2 defined in shell) |
| `$ bash` | (Start new shell) |
| `$ echo $FOO` | |
| `BAR` | (echoes value of FOO) |
| `$ echo $FOO2` | |
| | (empty) |

# Shell Scripting

- A script is simply a collection of commands that are intended to run as a group.
- Commands may or may not be dependent on each other.
- Variables, hence their values, can be transferred from one command to another.
- Supports complex choices and logic.
- A script is always executed in its own shell.

# Example Shell Script

- First example script: Hello world!

```
#!/bin/bash
# This is our first shell script!!
echo “Hello World!”
```

# Variables in Shell Scripting

- Variables are containers that store a value.
- All variables created in a script are shell variables.
- A script can access the environment variables in addition to its own shell variables.
- Variable can store any kind of value ie., string or integer or floating point number etc.

# Variables in Shell Scripting

```
INT=1
FLOAT=1.5
STR=hello
STR2=“hello world”
RND=asdf2341.sfe

echo $INT
echo “Value of FLOAT is $FLOAT”
echo “$STR is a string”
echo “$RND is non-sensical”
```

# Example Shell Script

- Second example script: `lsScr.sh`

```
#!/bin/bash
# List contents of scratch
cd $RCAC_SCRATCH
ls -l
```

- Make script executable, place it in PATH.

# Special shell variables

- Special Variables
  - $# = No. of parameters given to script
  - $@ = List of parameters given to script
  - $0 = Name of current program (script)
  - $1, $2.. = Parameter 1, 2 and so on..
  - $? = Exit value of last command run
- These variables are shell variables and only valid to the current shell.

# Even more special characters

- * matches every character, just as in regular expressions.
- So, ls *txt in a script will list all files whose name ends in txt.
- \ is an escape character which tells the shell to not interpret the character after it.
- \ is commonly used to escape the special characters such as *, $ etc.

# Example Shell Script

- Third example script: `lsScr.2.sh`

```
#!/bin/bash
# List contents of scratch
echo “Executing script : \”$0\” with $#
parameters”
cd $RCAC_SCRATCH
ls –l
```

- Make script executable, place it in PATH.

# Command Blocks

- Two fundamental blocks in scripting:
  - Loops
    Repeat the commands in the block until the exit condition is met.

  - Conditions
    Evaluate condition and if true execute commands in the block.

# Loops

- Two kinds of loops supported in bash:
  - for loop
    operates on a list and repeats commands in the block for each element on the list
  - while loop
    repeats commands in the block until an exit condition is met.

# for loops

- for loop
  operates on a list and repeats commands in the block for each element on the list

```
for x in [ list ];
do
      commands
done
```

# for loops

- for loop
  operates on a list and repeats commands in the block for each element on the list

```
for x in $( ls );
do
	echo “Found file $x”
done
```

# for loops

- for loop
  operates on a list and repeats commands in the block for each element on the list

```
for x in 1 2 3 4 5 6 7 8 9 10;
do
	echo “Value of x is : $x”
done
```

# while loops

- while loop
  repeats commands until exit condition is met

```
while condition;
do
        echo “Value of x is : $x”
done
```

# while loops

- while loop
  repeats commands until exit condition is met

```
x=10
while [ $x –gt 0 ];
do
	echo “Value of x is : $x”
	x=x-1
done
```

# Shell Scripting

- Condition blocks test for a condition and if TRUE execute one block and if FALSE execute another.

```
if [ condition ]
then
      Block 1
else
      Block 2
fi
```

# Shell Scripting

- Condition blocks test for a condition and if TRUE execute one block and if FALSE execute another.

```
x = 5
if [ $x –gt 0 ]
then
      echo “$x is divisible”
else
      echo “0 is not divisible”
fi
```

# breaking loops

- Break command asks the shell to exit the loop

```
x=10
while [ 1 ];
do
        echo “Value of x is : $x”
        x=x-1
        if [ $x == 0 ]
                break
done
```

# Run external commands

- backticks are a way to send a command to the shell and capture the result.
- It's a special character : `
- Eg.,

```
files = `ls *txt`
echo $files
```

# Functions in shell Scripting

- Functions separate logical blocks of code.
- Typically a function contains a piece of code that is used repeatedly in a script.
- Code in a function is only executed when a function is ”called”.
- We will cover functions in tomorrows lab section.