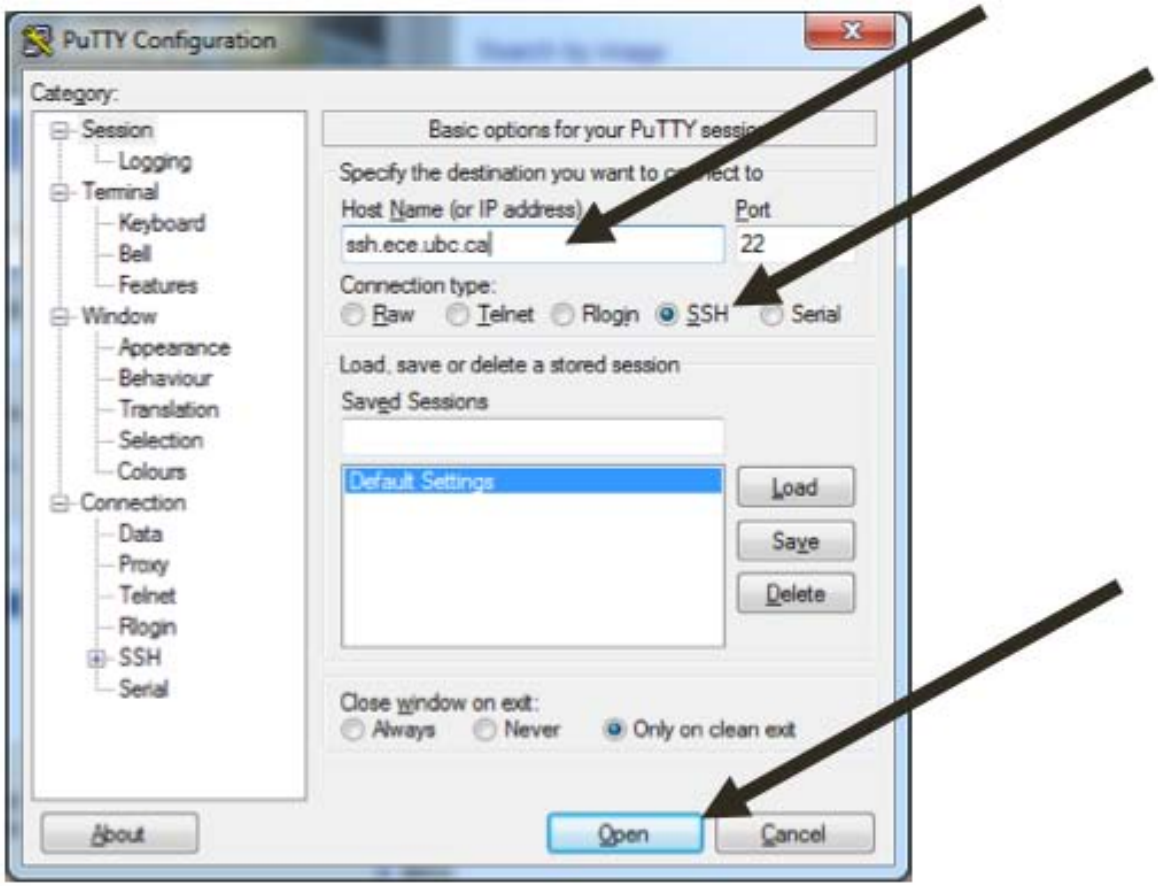
Linux Workshop II
by
Mehdi Karimi

March 2013

From a Windows Machine

- Download putty.exe www.putty.org



From a MAC

Open the terminal app

Type

ssh <username>@<server_name>

- Username: mkarimib
- Server name: ssh.ece.ubc.ca
- Example: ssh mkarimib@ssh.ece.ubc.ca

You can even connect remotely to your in-lab computer ….if you know the name of your computer on the ece network…you can do the following:

For example for me it's:

- ssh -L 3389:apt2:3389 mkarimib@ssh.ece.ubc.ca

  [-L [bind_address:]port:host:hostport]

- Dwld remote desktop connection and you can access your lab computer from home ☺
  - Remember to put 'local host' after dwlding.

Scripting in Linux:

- Most common scripting languages in Linux:
  - Perl
  - Python
  - TCL
  - Bash
- We go over some examples of each
- These are mainly used for test automation

Perl examples

- Take the following text and put it into a file called first.pl:
  - #!/usr/local/bin/perl

    print "Hi there!\n";
- Scalars, vectors, hashes
- scalars: $apple_count = 5;

  $count_report = "There are $apple_count apples.";

  print "The report is: $count_report\n”;

Perl examples

- $a = 5;
- $a++; # $a is now 6; we added 1 to it.
- $a += 10; # Now it's 16; we added 10.
- $a /= 2; # And divided it by 2, so it's 8.

Perl examples

- Arrays are lists of scalars. Array names begin with @. You define arrays by listing their contents in parentheses, separated by commas:

- @lotto_numbers = (1, 2, 3, 4, 5, 6);
- @months = ("July", "August”, "September");

Perl examples

- @months = ("July", "August", "September");
- print $months[0]; # This prints "July".
- $months[2] = “March"; # We just renamed September!
- If an array doesn't exist, by the way, you'll create it when you try to assign a value to one of its elements.
- $winter_months[0] = "December"; # This implicitly creates @winter_months.
- Arrays always return their contents in the same order; if you go through @months from beginning to end, no matter how many times you do it, you'll get back July, August and September in that order. If you want to find the length of an array, use the value $#array_name. This is one less than the number of elements in the array. If the array just doesn't exist or is empty, $#array_name is -1. If you want to resize an array, just change the value of $#array_name.

- @months = ("July", "August", "September");
- print $#months; # This prints 2.
- $a1 = $#autumn_months; # We don't have an @autumn_months, so this is -1.
- $#months = 0; # Now @months only contains "July".

Perl examples

- Hashes are called ``dictionaries'' in some programming languages, and that's what they are: a term and a definition, or in more correct language a key and a value. Each key in a hash has one and only one corresponding value. The name of a hash begins with a percentage sign, like % parents. You define hashes by comma-separated pairs of key and value, like so:

- %days_in_summer = ("July" => 31, "August" => 31, "September" => 30);
- You can fetch any value from a hash by referring to $hashname{key}, or modify it in place just like any other scalar.
- print $days_in_summer{"September"}; # 30, of course.
- $days_in_summer{"February"} = 29; # It's a leap year.
- If you want to see what keys are in a hash, you can use the keys function with the name of the hash. This returns a list containing all of the keys in the hash. The list isn't always in the same order, though; while we could count on @months to always return July, August, September in that order, keys %days_in_summer might return them in any order whatsoever.

Perl examples

- Loops: Almost every time you write a program, you'll need to use a loop. Loops allow you run a particular piece of code over and over again. This is part of a general concept in programming called flow control.
- Perl has several different functions that are useful for flow control, the most basic of which is for when you use the for function, you specify a variable that will be used for the loop index, and a list of values to loop over. Inside a pair of curly brackets, you put any code you want to run during the loop:
- for $i (1, 2, 3, 4, 5) {
  print "$i\n";
  }
- This loop prints the numbers 1 through 5, each on a separate line.

- A handy shortcut for defining loops is using .. to specify a range of numbers. You can write (1, 2, 3, 4, 5) as (1 .. 5). You can also use arrays and scalars in your loop list. Try this code and see what happens:

- @one_to_ten = (1 .. 10);
- $top_limit = 25;
- for $i (@one_to_ten, 15, 20 .. $top_limit) {
- print "$i\n";
- }
- The items in your loop list don't have to be numbers; you can use strings just as easily. If the hash %month_has contains names of months and the number of days in each month, you can use the keys function to step through them.

- for $i (keys %month_has) {
- print "$i has $month_has{$i} days.\n";
- }
- for $marx ('Groucho', 'Harpo', 'Zeppo', 'Karl') {

- print "$marx is my favorite Marx brother.\n";
- }

Perl examples

- The Miracle of Compound Interest
- You now know enough about Perl - variables, print, and for() - to write a small, useful program. Everyone loves money, so the first sample program is a compound-interest calculator. It will print a (somewhat) nicely formatted table showing the value of an investment over a number of years. (You can see the program at compound_interest.pl)
- The single most complex line in the program is this one:
- $interest = int (($apr / 100) * $nest_egg * 100) / 100;
- $apr / 100 is the interest rate, and ($apr / 100) * $nest_egg is the amount of interest earned in one year. This line uses the int() function, which returns the integer value of a scalar (its value after any fractional part has been stripped off). We use int() here because when you multiply, for example, 10925 by 9.25%, the result is 1010.5625, which we must round off to 1010.56. To do this, we multiply by 100, yielding 101056.25, use int() to throw away the leftover fraction, yielding 101056, and then divide by 100 again, so that the final result is 1010.56. Try stepping through this statement yourself to see just how we end up with the correct result, rounded to cents.

Perl examples

- Play Around!
- At this point you have some basic knowledge of Perl syntax and a few simple toys to play with - print, for(), keys(), and int(). Try writing some simple programs with them. Here are two suggestions, one simple and the other a little more complex:
- A word frequency counter. How often does each word show up in an array of words? Print out a report. (Hint: Use a hash to count of the number of appearances of each word.)
- Given a month and the day of the week that's the first of that month, print a calendar for the month. (Remember, you need \n to go to a new line.)
- http://www.perl.com/pub/2001/01/begperl6.html

Python examples

```
 print "Hello, World!”
str = "foo";
lst = ["abra", 2038, "cadabra”]
for char in str:
    print char
for elem in lst:
    print elem
def iterquad ():
    for i in range(5):
            yield (i*i)
for j in iterquad():
     print j
```

Python examples

- http://docs.python.org/2/tutorial/
- http://www.secnetix.de/olli/Python/

Tcl examples

```
% set l [list a b c]
a b c
% llength $l
3
% lappend l [list d e f]
a b c {d e f}
% llength $l
4
% lappend m [list a b c]
{a b c}
% llength $m
1
% lappend m [list d e f]
{a b c} {d e f}
% llength $m
2
set i 0
puts "i is set to $i"
set l [ list a b c]
puts "$l"
puts [llength $l]
puts [lindex $l 2]
```

```
for {set i 1} {$i < 10} {incr i 1} {
puts "i equals $i"
}


set sum 0
foreach value {1 2 3 4} {
set sum [expr $sum + $value]
}
puts $sum
10
```

```
>proc guess {value} {
global sum
if {$value < $sum} {
puts "too low"
} else {
if {$value > $sum} {
puts "too high"
} else { puts "you got it!"}
}
}
> guess 9
too low
```

Tcl examples

- Use Tcl while loop to copy a list from variable a to variable b, reversing the order of the elements along the way:

```
set b [list 1 3 4 5 6 6]
set a [list 2 4 5 2]
puts $b
puts $a
set i [expr  {[llength $a] - 1}]
while {$i>=0} {
lappend b [lindex $a $i]
set i [expr $i-1]
}
puts $b
```

Tcl examples

```
set days [list Monday Tuesday Wednesday Thursday \
    Friday Saturday Sunday]
set n [llength $days]

set i 0

while {$i < $n} {

    puts [lindex $days $i]
    incr i
}
```

```
#!/usr/bin/tclsh

proc maximum {x y} {

    if {$x > $y} {

        return $x
    } else {

        return $y
    }
}

set a 23
set b 32

set val [maximum $a $b]
puts "The max of $a, $b is $val"
```

Tcl examples

```
#!/usr/bin/tclsh

set names { John Mary Lenka Veronika Julia Robert }
set nums { 1 5 4 3 6 7 9 2 11 0 8 2 3 }

puts [lsort $names]
puts [lsort -ascii $names]
puts [lsort -ascii -decreasing $names]
puts [lsort -integer -increasing $nums]
puts [lsort -integer -decreasing $nums]
puts [lsort -integer -unique $nums]
```

```
#!/usr/bin/tclsh

proc power {a {b 2}} {
    if {$b == 2} {
        return [expr $a * $a]
    }

    set value 1

    for {set i 0} {$i<$b} {incr i} {

        set value [expr $value * $a]
    }

    return $value
}

set v1 [power 5]
set v2 [power 5 4]

puts "5^2 is $v1"
puts "5^4 is $v2"
```

Tcl examples

```
#!/usr/bin/tclsh

proc factorial n {
    if {$n==0} {
        return 1
    } else {
        return [expr $n * [factorial [expr $n - 1]]]
    }
}

# Stack limit between 800 and 1000 levels

puts [factorial 4]
puts [factorial 10]
puts [factorial 18]
```

```
#!/usr/bin/tclsh

proc power {a {b 2}} {
    if {$b == 2} {
        return [expr $a * $a]
    }

    set value 1

    for {set i 0} {$i<$b} {incr i} {
        set value [expr $value * $a]
    }

    return $value
}

set v1 [power 5]
set v2 [power 5 4]

puts "5^2 is $v1"
puts "5^4 is $v2"
```

Tcl examples

- In Tcl there can be nested lists; list in other lists.

- #!/usr/bin/tclsh

- set nums {1 2 {1 2 3 4} {{1 2} {3 4}} 3 4}

- puts [llength $nums]
- puts [llength [lindex $nums 2]]

- puts [lindex $nums 0]
- puts [lindex [lindex $nums 2] 1]
- puts [lindex [lindex [lindex $nums 3] 1] 1]

Bash examples

```
VAR1="$1”
VAR2="$2”
myvar = “hey”

if [ -z "$myvar" ]
then
      echo "myvar is not defined”
fi

if [ "$myvar" -eq 3 ]
then
   echo "myvar equals 3"
fi

if [ "$myvar" = "3" ]
then
   echo "myvar equals 3"
fi
```

```
#!/bin/bash

a=23           # Simple case
echo $a
b=$a
echo $b

# Now, getting a little bit fancier (command substitution).

a=`echo Hello!`   # Assigns result of 'echo' command to 'a' ...
echo $a
#  Note that including an exclamation mark (!) within a
#+ command substitution construct will not work from the command-line,
#+ since this triggers the Bash "history mechanism."
#  Inside a script, however, the history functions are disabled.

a=`ls -l`       # Assigns result of 'ls -l' command to 'a'
echo $a         # Unquoted, however, it removes tabs and newlines.
echo
echo "$a"       # The quoted variable preserves whitespace.
                # (See the chapter on "Quoting.")

exit 0
```

Bash examples

Common Bash comparisons

Operator	Meaning	Example
-z	Zero-length string	[-z "$myvar"]
-n	Non-zero-length string	[-n "$myvar"]
=	String equality	["abc" = "$myvar"]
!=	String inequality	["abc" != "$myvar"]
-eq	Numeric equality	[3 -eq "$myinteger"]
-ne	Numeric inequality	[3 -ne "$myinteger"]
-lt	Numeric strict less than	[3 -lt "$myinteger"]
-le	Numeric less than or equals	[3 -le "$myinteger"]
-gt	Numeric strict greater than	[3 -gt "$myinteger"]
-ge	Numeric greater than or equals	[3 -ge "$myinteger"]
-f	Exists and is regular file	[-f "$myfile"]
-d	Exists and is directory	[-d "$mydir"]
-nt	First file is newer than second one	["$myfile" -nt ~/.bashrc]
-ot	First file is older than second one	["$myfile" -ot ~/.bashrc]

Bash examples

- #!/usr/bin/env bash

- for x in one two three four
- do
- echo number $x
- done

- output:
- number one
- number two
- number three
- number four

- #!/bin/bash

- if [-d "work"]
- then
- rm -rf work
- mkdir work
- cd work
- if [! -d "sal2"]
- then
- mkdir sal2
- fi
- fi
- cd ..
- ////////////////
- tar xzf /usr/src/distfiles/sed-3.02.tar.gz
- cd sed-3.02
- ./configure --prefix=/usr
- make

Other good-to-knows…

- http://ctg.ece.ubc.ca/wiki/doku.php?id=linux_tricks

- .snapshot

Yet I'm learning Linux on my every day work routines….

- This was just the beginning, but the ones we just reviewed were the most common commands we need to know all the time….
- You may google
  - for piping command to do multiple commands at a time in a single command line
  - learn some other commands like xargs, touch, tail –l, awk, etc.
  - aliases

Thanks for coming to our workshop.