CS 35L Discussion 1A Week 1

More in Linux and Shell Scripting

Reminds

- Assignment 1!
  - 4 Oct 2022 - **11:55** pm UCLA Time
- Midterm
  - **24 Oct** 2022 - **In-class** midterm
  - The date on class website will be corrected soon

Last Week

1. Environment Setup for Assignments
   a. Connect to the SEASNet
2. Linux Basics
   a. The Unix File System
   b. Command to navigate through the file system
3. Some Useful Shell Commands/Tricks
   a. `man` command
   b. Other useful commands
   c. `chmod` command

Contents

1. Emacs -Why?

- Editor
  - VSCode, Sublime, notepad++ ...
  - Emacs, Vim ...
  - All of them:
    - Provide some highlight for files in different format
    - Provide some commands for easier editing
  - So why Emacs??
    - Why bother with those complex commands!!

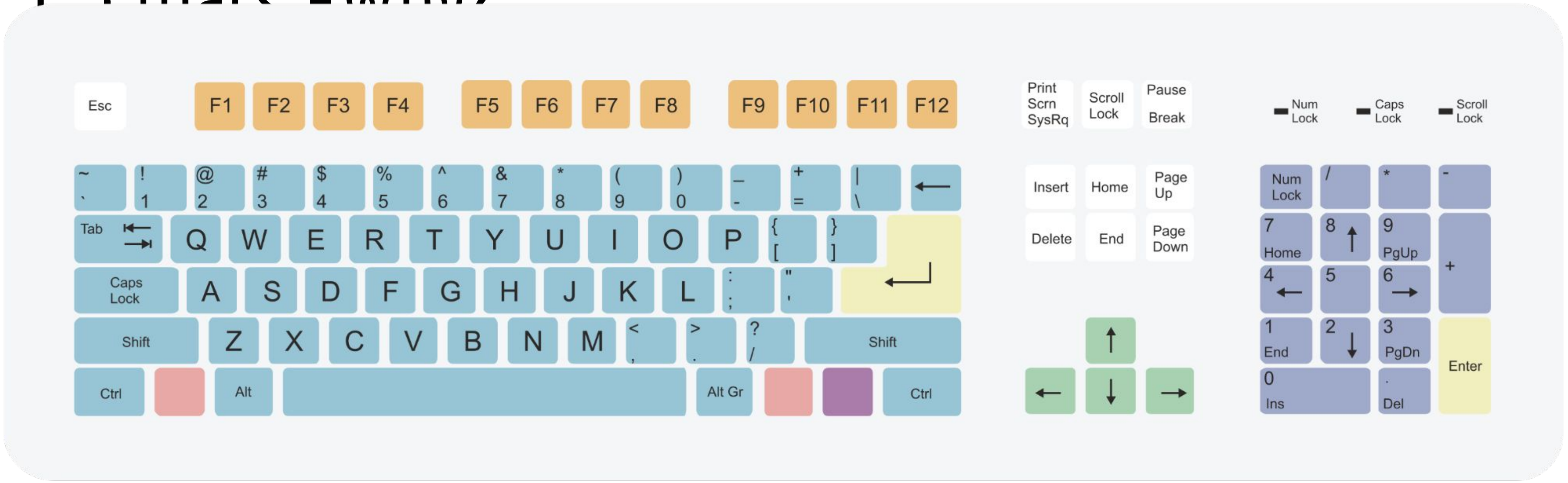
1. Emacs -Why?

- Sometimes we have no choice
  - No GUI
  - No mouse
  - Only terminal + keyboard

1. Emacs - Why?

- Sometimes we have no choice
  - No GUI
  - No mouse
  - Only terminal + keyboard
- For better efficiency

1. Emacs -Why?



🟦 Typewriter keys

🟧 Function keys

🟨 Enter keys

🟥 System keys

🟦 Numeric keypad

⬜ Other

🟪 Application key

🟩 Cursor control keys

1. Emacs -Why?

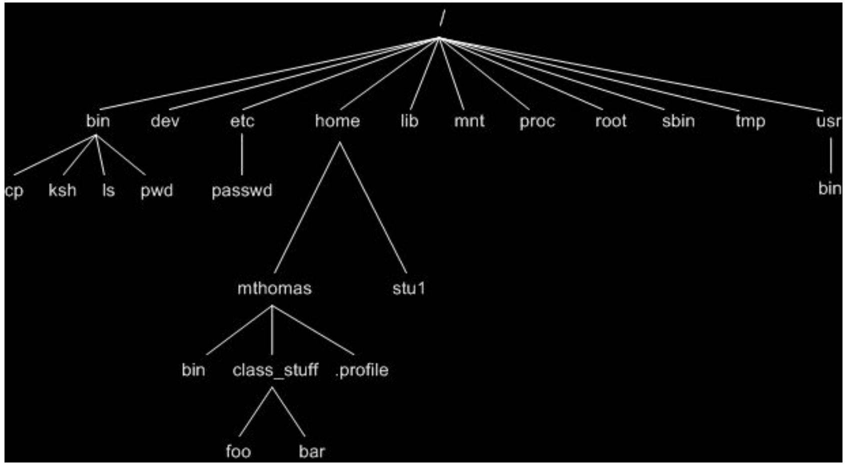
- Sometimes we have no choice
  - No GUI
  - No mouse
  - Only terminal + keyboard
- For better efficiency
  - Focus your hand on the left part of your keyboard
    - Increase the max speed limit of typing
  - Move cursors in almost anyway you want
    - Better functionality compared to arrow keys
- Other advantages
  - Link
- Editor is to make our life easier but not harder!
- Our goal is always: easy + fast!

Contents

2. Last Week - Unix File System

- The Unix File System: Tree structure
  - *bin*: short for binaries; the directory for commonly used executable commands
  - *home*: contains user directories and files
- Navigate through the system

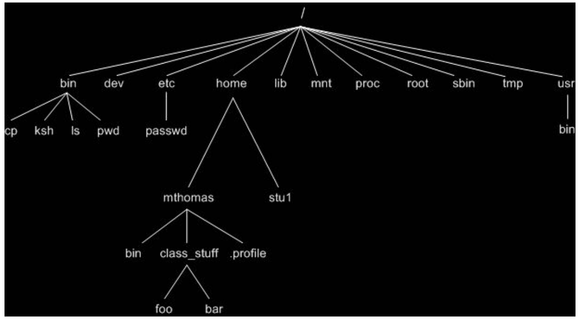
pwd	Print working directory
ls [directory]	List directory contents; -l for long format; -a for list all ...
cd [directory]	Change directory
.	Current directory
..	Parent directory
mkdir [directory]	Make a new directory
touch [file]	Create a file
rm [file] | rm -r [directory]	Remove a file / directory
cp [source] [destination]	Copy files; Copy directories (with -r)
mv [source] [destination]	Move/rename a file

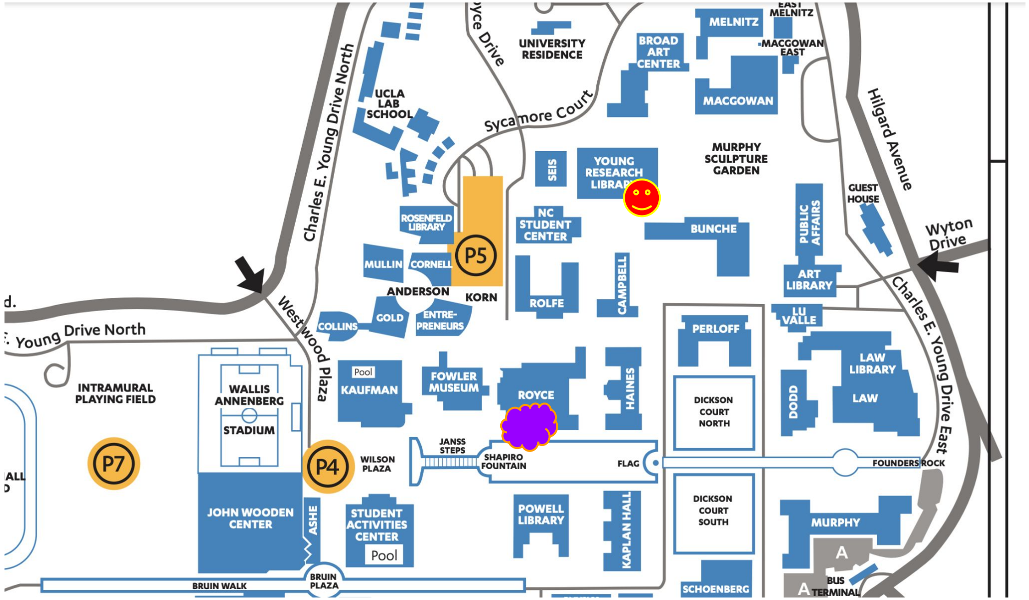


https://homepages.uc.edu/~thomam/Intro_Unix_Text/File_System.html

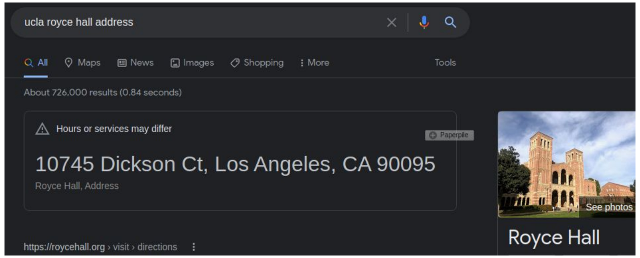
2. Path

- Locate a file/folder in your File System





- Absolute Address



ucla royce hall address

All · Maps · News · Images · Shopping · More · Tools

About 726,000 results (0.84 seconds)

Hours or services may differ

10745 Dickson Ct, Los Angeles, CA 90095

Royce Hall, Address

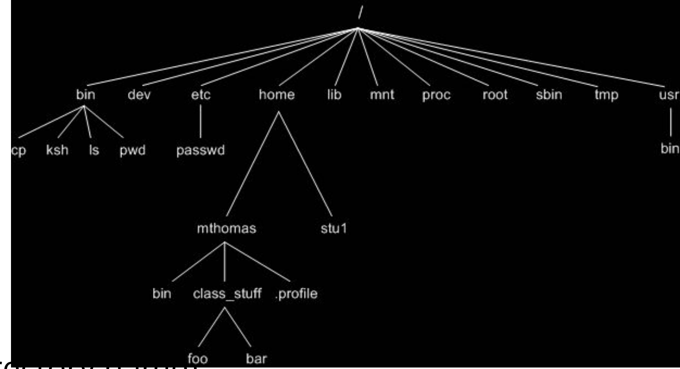
https://roycehall.org › visit › directions

Royce Hall

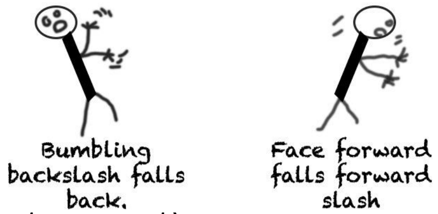
- Relative Way
  - Start from Young Research Lib
  - Head South for xx miles
  - Head West for xxx miles
  - (Go straight, turn right…)

https://homepages.uc.edu/~thomam/Intro_Unix_Text/File_System.html

2. Absolute / Relative Path



- Path
  - Unique location to a file or a folder in a file system
  - A combination of / and alphanumeric characters (/ after every directory name)
  - E.g.: "`/usr/bin`" "`/home/mthomas`" "`./class_stuff/foo`"
- Absolute path
  - Always starts from the root directory "/"
  - E.g.: "`/usr/bin`" "`/home/mthomas/class_stuff/foo`"
- Relative path
  - The path related to the present working directory (the output of "pwd", default starting point)
  - Never starts with a "/"
  - There are "infinite" number of relative paths to a file
  - Use .(the current directory) and ..(parent directory)
  - E.g.: (current dir: `/home/mthomas/`) goto "`class_stuff/foo`" : **`cd class_stuff/foo`**
  - E.g.: (current dir: `/home/mthomas/class_stuff/foo`) goto "`../../../stu1`": **`cd ../../../stu1`**
- Other tricks
  - `~` : home directory (the directory when you first login, in this case `/home/mthomas/`)
    - (e.g.: "`cd ~`" "`cd ~username`")
  - `-` : the last directory you just visited


Bumbling backslash falls back.
Face forward falls forward slash

Contents

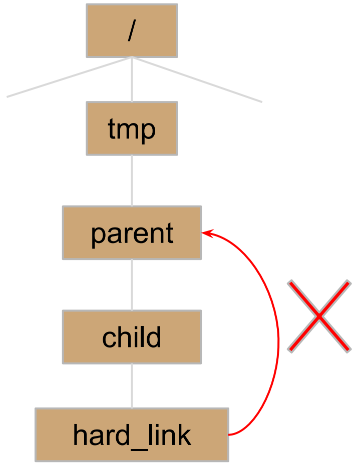
3. Soft and Hard Links

- Link (Two types: Hard link v.s Soft link)
    - A pointer to a file
    - Allow more than one file name to refer to the same file
- Inode:
    - A **data structure** in a Unix file system that describes a **file-system object** such as a **file** or a **directory**
    - Stores the disk block locations of the object's data, and attributes (ownership, access, times of last change, ...)
    - Identified by an integer (i.e. **i-number** or **inode number**)
    - Show the inode number index: `ls -i`
- Hard link v.s Soft link (Symbolic link)
    - Basic difference:
        - Hard link file: the ***same*** Inode (index node) value as the original
        - Soft link file: ***separate*** Inode value, stores a path string that points to the original file

3. Soft and Hard Links -- Difference

- Hard link v.s Soft link (Symbolic link)
  - Basic difference:
    - Hard link file: the ***same*** Inode (index node) value as the original
    - Soft link file: ***separate*** inode value that points to the original file
  - Behave differently when the source of the link is moved or removed
    - Hard link:
      - Always refer to the source, even if moved or removed
      - Increase the reference count of a location in memory
    - Symbolic link:
      - Not updated (merely contain a string which is the pathname of its target)
      - Work as a shortcut (like in Windows), stores a path string
  - Link to directories
    - Next slide

3. Soft and Hard Links -- For Dirs



- Hard link v.s Soft link (Symbolic link)
  - Link to directories
    - Hard link:
      - We CANNOT do that to avoid recursive loops
      - Unix file systems are tree-structured
      - Suppose we can create hard link to directories
        - `“cd /tmp/parent/child/”; “ln /tmp/parent hard_link”`
        - `Recursive loop and ambiguity! /tmp/parent → /tmp/parent/child/ → /tmp/parent`
    - Soft link:
      - Can link to a directory
      - Just a shortcut **string**
  - Command to create hard/soft links:
    - Hard link: `ln [original_filename] [hard_link_name]`
    - Soft link: `ln -s [original_filename] [soft_link_name]`

Contents

4. Shell Operators

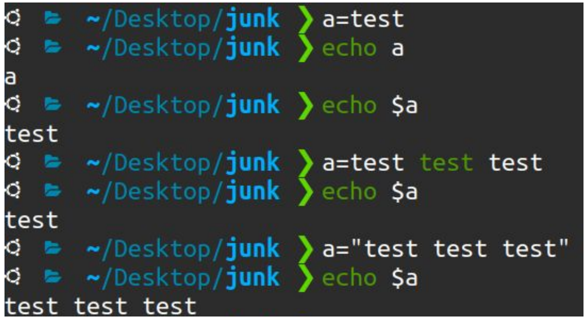
- Pipe operator `|`: passes the output of one command as input to another
  - E.g.: `ls . | grep “a”`
  - More control operators: https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap03.html#tag_03_113
- Redirection operator: https://www.gnu.org/savannah-checkouts/gnu/bash/manual/bash.html#Redirections
  - **stdin**, **stdout** and **stderr**: data streams created when you launch a Linux command
    - 0: stdin, 1: stdout, 2:stderr
  - `command <` : gives input to a command, e.g.: `grep "int" -n < hello.c`
  - `command > out.txt` : Directs the output of a command into a file out.txt
  - `E.g.: command 0<&- >out.txt 2>error.txt`
    - `# turn off the stdin, redirect the stdout to out.txt, stderr to error.txt`
  - `>` v.s. `>>`:
    - `>` : overwrite the file (erase previous contents)
    - `>>` : append to the file (preserve contents, write new things to the end of the file)

Contents

5. Shell Script

- Shell script
  - Create a file
    - With extension `.sh` (not required)
  - Add 1st line `#!/bin/bash` or `#!/usr/bin/bash`
  - Edit your script
  - Add execute permission with `chmod +x myscript`
  - Run your script!
    - `[dir]/[script]` E.g.: `./filename.sh`
- `#!` In the first line
  - Tell the shell what program to interpret the script with, when executed
  - When the shell runs a program, it asks the kernel to start a new process and run the given program in that process. It knows how to do this for compiled programs. But for a script, we should tell the OS how to run the file. With `#!/usr/bin/bash`, the shell knows to use the bash interpreter to run the file.
  - Normally `#` starts a comment line

5. Shell Script -- Variables

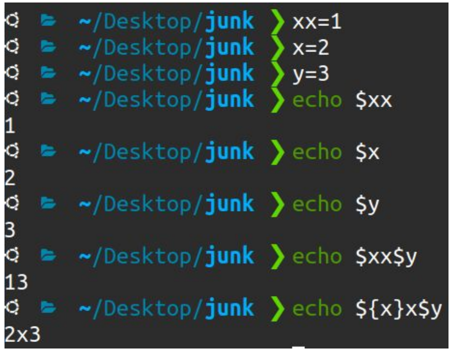


```
~/Desktop/junk ❯ a=test
~/Desktop/junk ❯ echo a
a
~/Desktop/junk ❯ echo $a
test
~/Desktop/junk ❯ a=test test test
~/Desktop/junk ❯ echo $a
test
~/Desktop/junk ❯ a="test test test"
~/Desktop/junk ❯ echo $a
test test test
```

- Shell variables
  - Assign variables by assignments
    - `a=1`
    - `a=test #no space between =`
    - `a=”test test test” #we want our variable to contain whitespaces, we need to use quotes`
  - Show values of a variable
    - `echo $a (“echo a” will not work)`
  - Shell variables disappear once log off, they are specified to the current session

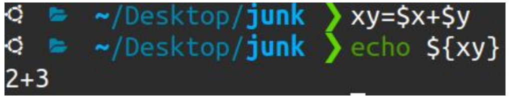
5. Shell Script -- Variables Continue

- Shell variables
  - Parameter Expansion
    - `$x` and `${x}` are mostly equivalent
    - `{}` gives less ambiguity
    - `$xx$y` and `${x}x$y` are not the same

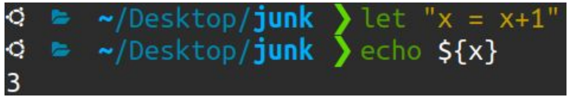


```
~/Desktop/junk ❯ xx=1
~/Desktop/junk ❯ x=2
~/Desktop/junk ❯ y=3
~/Desktop/junk ❯ echo $xx
1
~/Desktop/junk ❯ echo $x
2
~/Desktop/junk ❯ echo $y
3
~/Desktop/junk ❯ echo $xx$y
13
~/Desktop/junk ❯ echo ${x}x$y
2x3
```

  - Type of your variables
    - Bash variables are **untyped**
    - No need to declare type for bash variables
    - Default type: **string**
    - Depending **on the context**, **arithmetic** operations and comparisons are allowed



```
~/Desktop/junk ❯ xy=$x+$y
~/Desktop/junk ❯ echo ${xy}
2+3
```

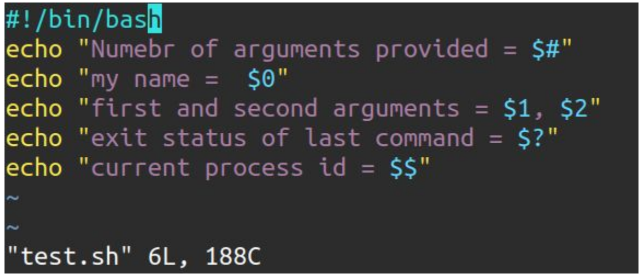


```
~/Desktop/junk ❯ let "x = x+1"
~/Desktop/junk ❯ echo ${x}
3
```

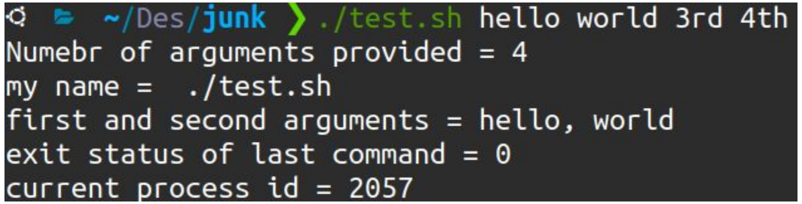
5. Shell Script -- Built-in Variables

- Shell variables
    - Built-in shell variables (can be accessed in the **shell script**)

$#	Number of arguments provided to script
$0	Name of script
$1, $2, etc	1st and 2nd argument, etc
${15}, ${23}, etc	For arguments greater than 9
$?	Exit status of last command
$$	Current running process ID



```
#!/bin/bash
echo "Numebr of arguments provided = $#"
echo "my name =  $0"
echo "first and second arguments = $1, $2"
echo "exit status of last command = $?"
echo "current process id = $$"
~
~
"test.sh" 6L, 188C
```

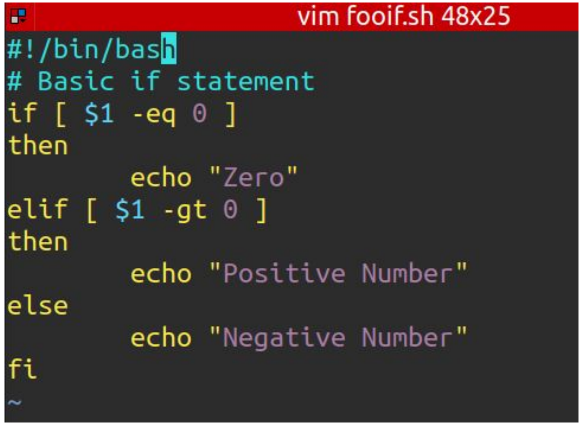


```
~/Des/junk ❯ ./test.sh hello world 3rd 4th
Numebr of arguments provided = 4
my name =  ./test.sh
first and second arguments = hello, world
exit status of last command = 0
current process id = 2057
```

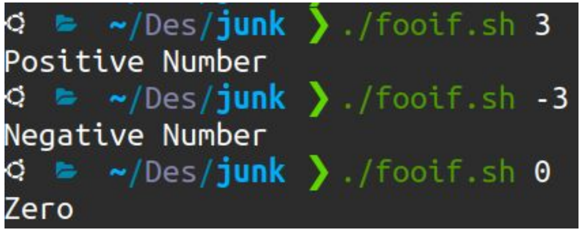
5. Shell Script -- If Statement

- if Statement

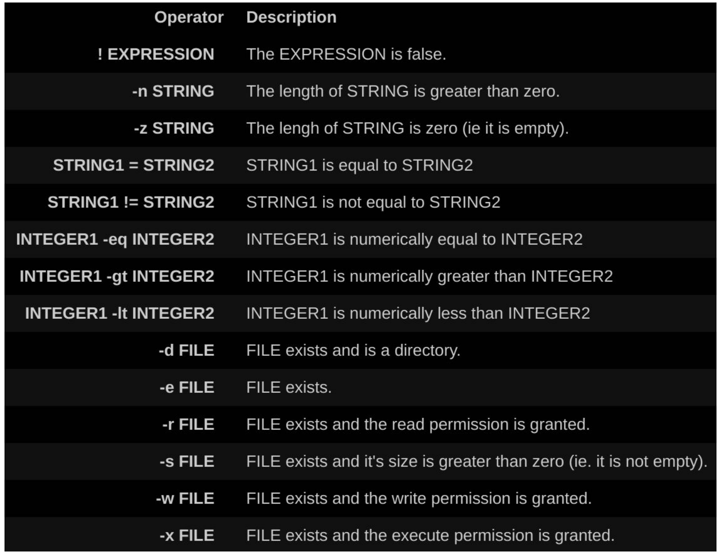
```
if [conditions]
then
        [commands]
elif [conditions]
then
        [commands]
elif [conditions]
then
        [commands]
...
else
        [commands]
fi
```



```
vim fooif.sh 48x25
#!/bin/bash
# Basic if statement
if [ $1 -eq 0 ]
then
        echo "Zero"
elif [ $1 -gt 0 ]
then
        echo "Positive Number"
else
        echo "Negative Number"
fi
~
```



```
~/Des/junk ❯ ./fooif.sh 3
Positive Number
~/Des/junk ❯ ./fooif.sh -3
Negative Number
~/Des/junk ❯ ./fooif.sh 0
Zero
```



Operator	Description
! EXPRESSION	The EXPRESSION is false.
-n STRING	The length of STRING is greater than zero.
-z STRING	The lengh of STRING is zero (ie it is empty).
STRING1 = STRING2	STRING1 is equal to STRING2
STRING1 != STRING2	STRING1 is not equal to STRING2
INTEGER1 -eq INTEGER2	INTEGER1 is numerically equal to INTEGER2
INTEGER1 -gt INTEGER2	INTEGER1 is numerically greater than INTEGER2
INTEGER1 -lt INTEGER2	INTEGER1 is numerically less than INTEGER2
-d FILE	FILE exists and is a directory.
-e FILE	FILE exists.
-r FILE	FILE exists and the read permission is granted.
-s FILE	FILE exists and it's size is greater than zero (ie. it is not empty).
-w FILE	FILE exists and the write permission is granted.
-x FILE	FILE exists and the execute permission is granted.

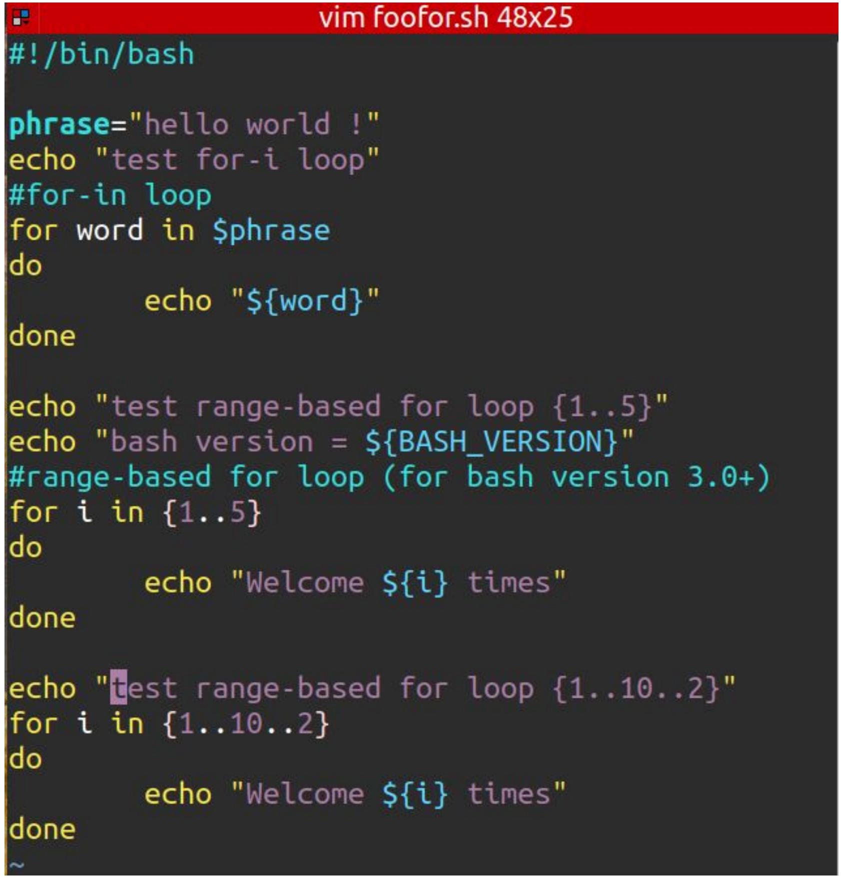
Reference: https://tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_01.html

5. Shell Script -- For Loop

- For Loop

```
for [xx] in [xxx]
do
        [commands]
done
```

  - Range-based for loop
    - Works for bash version 3.0+
    - `for i in {start .. end .. increment}`
      - `Default increment: 1`

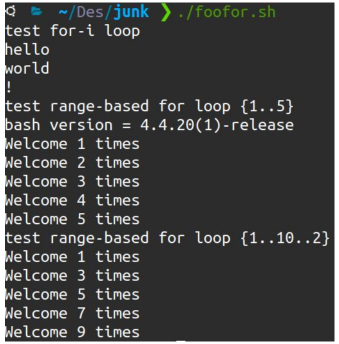


```
#!/bin/bash

phrase="hello world !"
echo "test for-i loop"
#for-in loop
for word in $phrase
do
        echo "${word}"
done

echo "test range-based for loop {1..5}"
echo "bash version = ${BASH_VERSION}"
#range-based for loop (for bash version 3.0+)
for i in {1..5}
do
        echo "Welcome ${i} times"
done

echo "test range-based for loop {1..10..2}"
for i in {1..10..2}
do
        echo "Welcome ${i} times"
done
~
```



```
~/Des/junk ❯ ./foofor.sh
test for-i loop
hello
world
!
test range-based for loop {1..5}
bash version = 4.4.20(1)-release
Welcome 1 times
Welcome 2 times
Welcome 3 times
Welcome 4 times
Welcome 5 times
test range-based for loop {1..10..2}
Welcome 1 times
Welcome 3 times
Welcome 5 times
Welcome 7 times
Welcome 9 times
```

Reminds

- Assignment 1!
  - 4 Oct 2022 - **11:55** pm UCLA Time
- Midterm
  - **24 Oct** 2022 - **In-class** midterm
  - The date on class website will be corrected soon