CS 97 - Discussion 1F
Week 2

More in Linux, Shell Scripting and Our Project

Reminds

- Assignment 1 is due today!
  - Jan 15 2021
  - **11:55** pm UCLA Time
  - Submission:
    - `lab1.drib` and any later dribble files that you generate.
    - The `hello-??` files of Lab 1.6.
    - `myspell`
    - `notes.txt`, a text file containing any other notes or comments that you'd like us to see

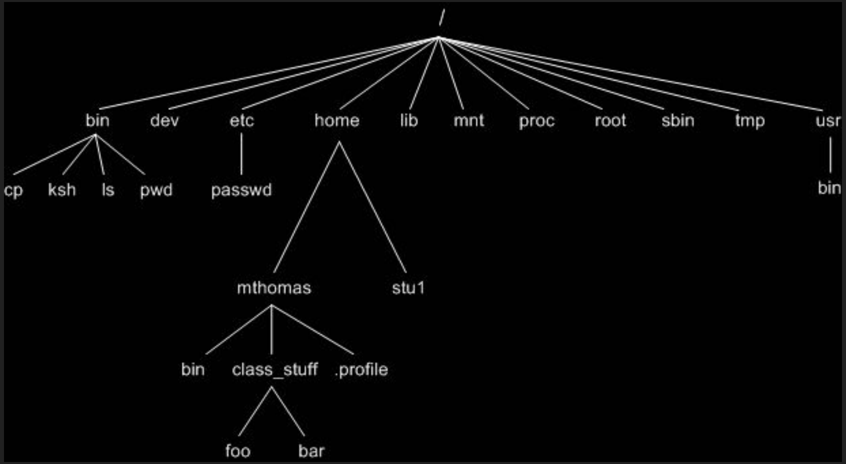
Contents

1. **Absolute / Relative Path**
2. Soft / Hard Links
3. Shell Script
4. Project (Group)

1. Last Week - Unix File System

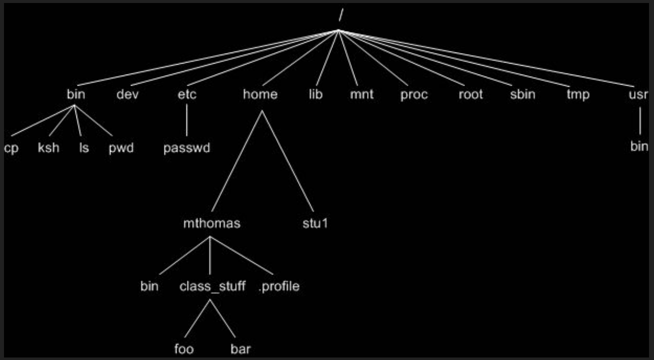
- The Unix File System: Tree structure
  - *bin*: short for binaries; the directory for commonly used executable commands
  - *home*: contains user directories and files
- Navigate through the system

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

1. 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/class_stuff/foo`) "`../../../stu1`"
- Other tricks
  - `~` : home directory (the directory when you first login) (e.g.: "`cd ~`" "`cd ~username`")
  - `-` : the last directory you just visited

Contents

1. Absolute / Relative Path
2. **Soft / Hard Links**
3. Shell Script
4. Project (Group)

2. Soft and Hard Links

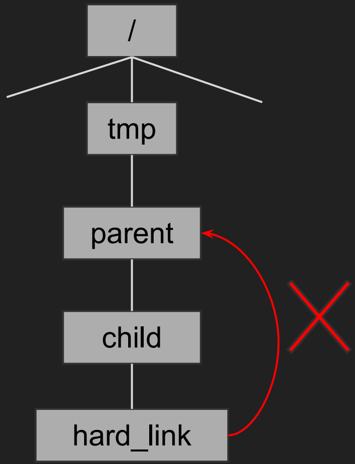
- Link
  - A pointer to a file
  - Allow more than one file name to refer to the same file
- Hard link v.s Soft link (Symbolic link)
  - Basic difference:
    - Hard link file: the ***same*** Inode (index node) value as the original
    - Soft link file: ***separate*** Inode value that points to the original file
    - 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(access, times of last change, ...)
      - Show the inode number index: `ls -i`

2. Soft and Hard Links -- Continue

- 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)
  - Link to directories
    - Next slide

2. Soft and Hard Links -- Continue

- 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

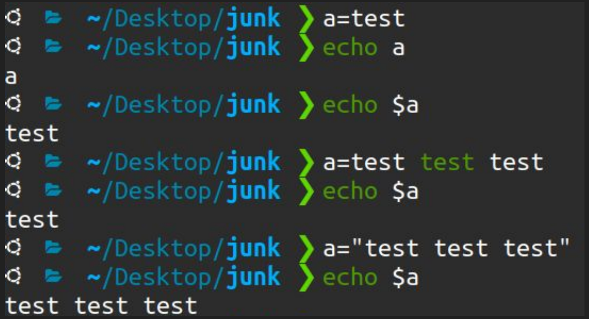
1. Absolute / Relative Path
2. Soft / Hard Links
3. **Shell Script**
4. Project (Group)

3. Shell Script

- Create a file
  - With extension `.sh` (not required)
- Add 1st line `#!/bin/bash` or `#!/usr/bin/bash`
  - Tell the shell what program to interpret the script with, when executed
- Edit your script
- Add execute permission
  - `chmod +x myscript`
- Run your script!
  - `[dir]/[script]`
  - E.g.: `./script.sh`

3. Shell Script

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

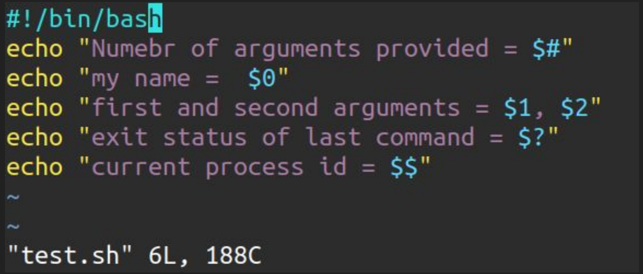


```
~/Desktop/junk ❯ a=test
~/Desktop/junk ❯ echo a
a
~/Desktop/junk ❯ echo $a
test
~/Desktop/junk ❯ a=test test test
~/Desktop/junk ❯ echo $a
test
~/Desktop/junk ❯ a="test test test"
~/Desktop/junk ❯ echo $a
test test test
```

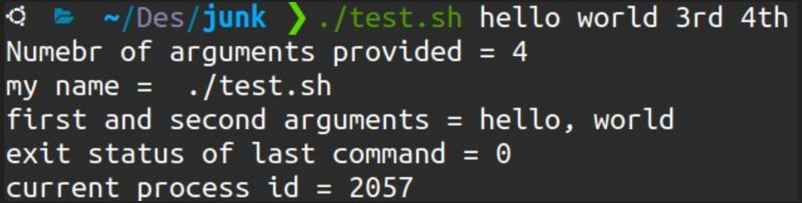
3. Shell Script

- 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



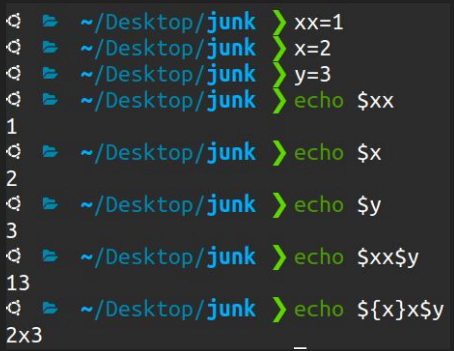
```
#!/bin/bash
echo "Numebr of arguments provided = $#"
echo "my name =  $0"
echo "first and second arguments = $1, $2"
echo "exit status of last command = $?"
echo "current process id = $$"
~
~
"test.sh" 6L, 188C
```



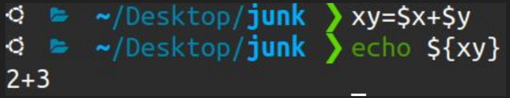
```
~/Des/junk ❯ ./test.sh hello world 3rd 4th
Numebr of arguments provided = 4
my name =  ./test.sh
first and second arguments = hello, world
exit status of last command = 0
current process id = 2057
```

3. Shell Script

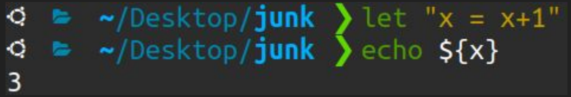
- Shell variables
  - Parameter Expansion
    - `$x` and `${x}` are mostly equivalent
    - `{}` gives less ambiguity
    - `$xx$y` and `${x}x$y` are not the same
  - Type of your variables
    - Bash variables are untyped
    - No need to declare type for bash variables
    - Default type: string
    - Depending on the context, arithmetic operations and comparisons are allowed



```
~/Desktop/junk ❯ 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
```



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



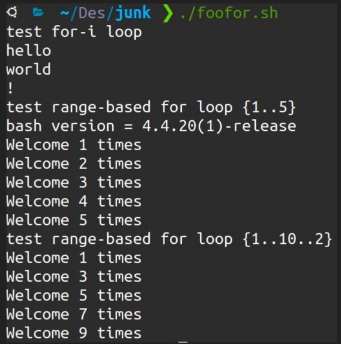
```
~/Desktop/junk ❯ let "x = x+1"
~/Desktop/junk ❯ echo ${x}
3
```

3. Shell Script

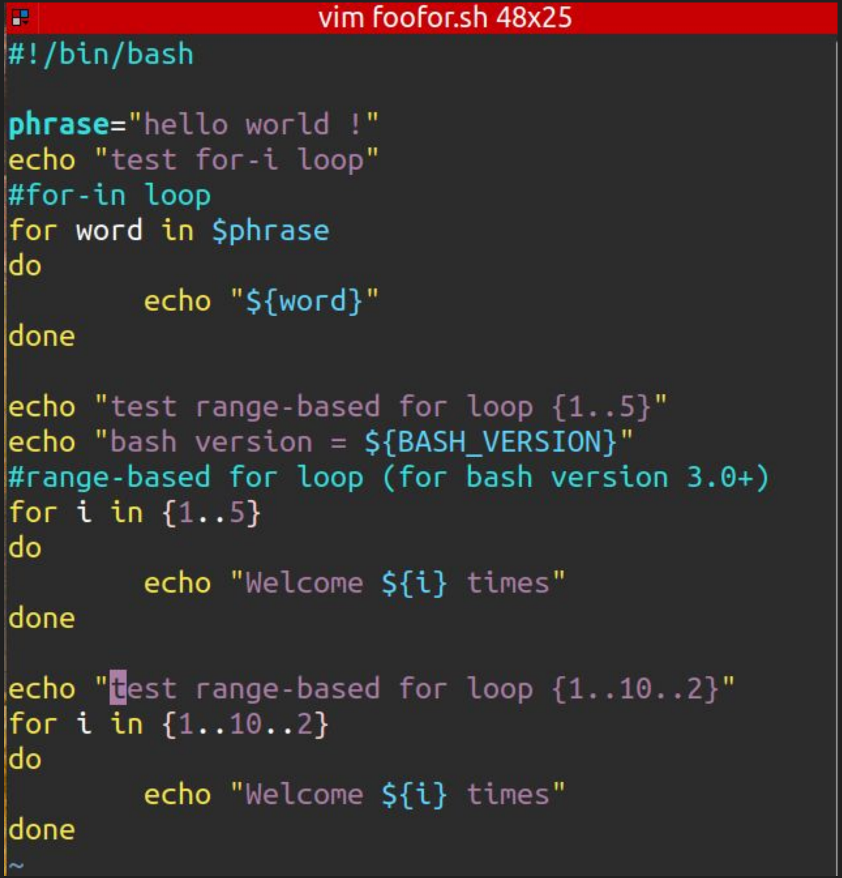
- For Loop

```
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`



```
~/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
```



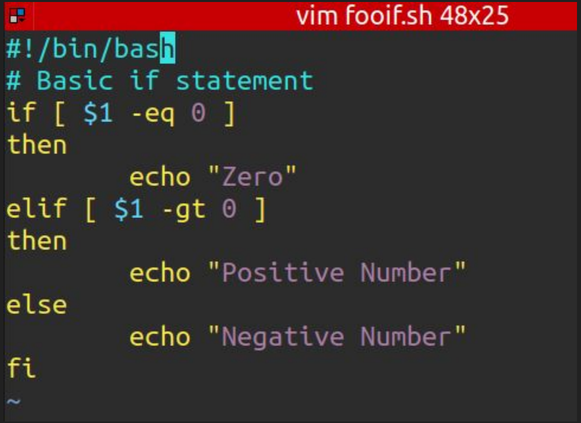
vim foofor.sh 48x25

```
#!/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
~
```

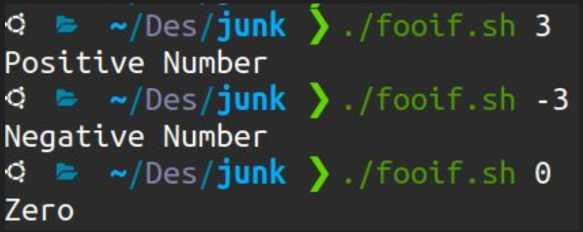
3. Shell Script

- if Statement

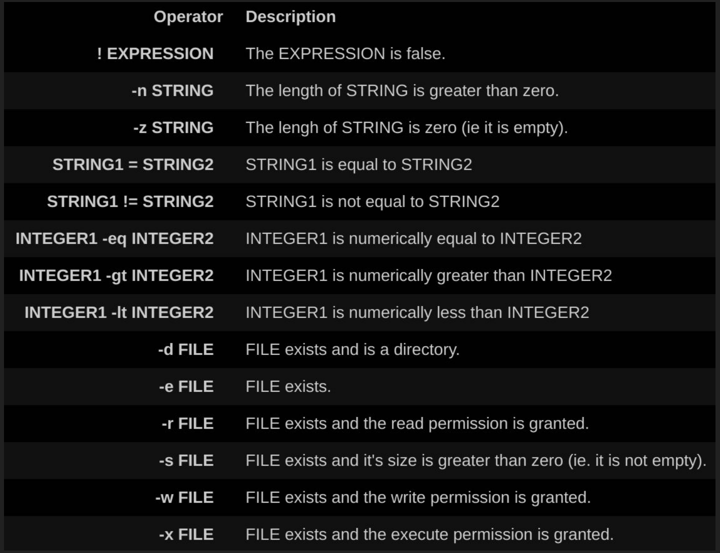
```
if [conditions]
then
        [commands]
elif [conditions]
then
        [commands]
elif [conditions]
then
        [commands]
...
else
        [commands]
fi
```



```
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

Contents

1. Absolute / Relative Path
2. Soft / Hard Links
3. Shell Script
4. **Project (Group)**

4. Project (Group)

- Open-ended
  - Feel free to choose any ideas!
  - Great for your resume
- Key requirements
  - Some type of Client-Server Application
    - Front-end Tech
    - Back-end Tech
  - Applications should support:
    - Dynamic data, website updates based on what is sent back and forth to the server
    - Uploading: Client upload persistent data to server
    - Searching: Can search through server-side data
    - 3 more unique features, based on your project idea

4. Project (Group)

- Find your group!
    - Sign-up sheet: https://docs.google.com/spreadsheets/d/1hURVny1igUp4yw2P9y-jczevA2VNisy1tHDuIWo0E8c/edit?usp=sharing
    - If you already have a group of 2 or more, you can sign yourself up as a group with your preferred working timezone.
    - If you are an individual
        - Reach out to groups you are interested in working with
        - Or just put yourself and your preferred time zone on the right-hand side, we will randomly allocate these students at the end

- Initial Project Proposal is due **Friday Week 4**

4. Project (Group)

- Examples!