Introduction to Linux

Research Computing Core Group (RCCG)

Office of the Vice President for Research (OVPR)

February, 2026

جامعة
حمد بن خليفة
HAMAD BIN KHALIFA
UNIVERSITY

عضو في مؤسسة قطر
Member of Qatar Foundation

Acknowledgements

- Most examples, scripts, and some of the diagrams in this short course have been adapted from “Unix Shells By Example (3rd Edition)” by Ellie Quigley.

- Some example scripts used in the slides were written by the staff of RCCG

- Any additional source material will be cited in the subsections in which it is used.

NOTE: This course originally spanned 3 consecutive sessions. Subsequently, less relevant content has been removed from the slides. Often, entire slides were removed, but in some cases some content on the included slides was greyed out. Readers may safely ignore this material to stay focused; but if they wish, they may avail the additional details as well.

Overview

- Session 1
  - Remote access to Linux systems
  - Unix command format
  - Basic commands
  - Filesystem concepts
  - Regular expressions
- Session 2
  - I/O redirection
  - The process hierarchy
  - Shell features, variables, quoting

For More Help…

Website: https://rccg.hbku.edu.qa/

Email: support@hbku.edu.qa

- Your queries should include, when possible, the following items:
  - Informative e-mail (subject) headers
  - The name of the system you are working on (e.g. “raad2”, “hazeem”)
  - Relevant error messages
  - Location of relevant files: job files, error files, etc.
  - Which program/package you were running
  - Clear description of the problem

  - E.g. : "Subject: raad2 MATLAB job out of memory error"

Remote Access Software

- MobaXterm is a free remote access software for Windows systems. The “Home Edition” is free for personal use.

http://mobaxterm.mobatek.net/download-home-edition.html

- If you don’t have admin privileges on your PC, download the “portable” version.

- MacOS/Linux users can use the built-in ‘Terminal’ application

Your Account Credentials

- You will normally deal with two sets of credentials
  - "domain" username & password (e.g. "itamboli")
  - supercomputer username and password (e.g. "itambol89")
- When outside the HBKU building, you must first establish a VPN connection to our network with your domain credentials.
- If you are in the building, or already connected via VPN, you may then use the supercomputer credentials to log in to the system.
- New users are forced to change their initial temporary domain password upon first login. The domain password then expires one year after that password change.
- New users are also forced to change their initial temporary supercomputer password. This too expires after one year.

Exercise

- Use MobaXterm/Terminal application to log in to raad2
- Explore some basic features of the interface
- Make sure you can start the “gedit” program on raad2

Linux Commands

Anatomy of a UNIX Command

Example 1

```
$ ls -l --time=access myDirectory
```

Example 2

```
$ ls -ltr
```

Command name

Options

- Leading single dash
  - This style of option can be "combined"
  - Can take option arguments (w/out = sign)
- Leading double dashes
  - Cannot be combined
  - Can take option arguments (w/ = sign)

Command arguments

How to Use the Online “Man”ual

- A *man page* is a special ASCII (text) help file
- Most commands have their own man page
- Accessed using the **`man`** command
- The layout of a man page follows certain conventions
  - Each man page is assigned a section # (of a virtual UNIX manual) to which it belongs
  - Each page itself is divided into sections
- Some man “pages” may be very lengthy. Certain keystrokes can help navigation in such cases.
- **`man intro`** provides a nice introduction to Linux.

Getting Help with ‘man’

If we typed **`man ls`** at the command line, we would see…

```
LS(1)                          User Commands                          LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about  the  FILEs  (the current directory by default).  Sort entries
       alphabetically if none of -cftuvSUX nor --sort.

       Mandatory arguments to long options are mandatory for short options too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..

       --author
              with -l, print the author of each file

       -b, --escape
              print octal escapes for nongraphic characters

           .
           .
           .
           .
```

Getting Help with ‘man’

If we continued to scroll through the output…

```
            .
            .
            .

 Exit status:
       0      if OK,

       1      if minor problems (e.g., cannot access subdirectory),

       2      if serious trouble (e.g., cannot access command-line argument).

AUTHOR
       Written by Richard M. Stallman and David MacKenzie.

REPORTING BUGS
       Report ls bugs to bug-coreutils@gnu.org
       GNU coreutils home page: <http://www.gnu.org/software/coreutils/>
       General help using GNU software: <http://www.gnu.org/gethelp/>
       Report ls translation bugs to <http://translationproject.org/team/>

COPYRIGHT
       Copyright © 2010 Free Software Foundation, Inc.  License GPLv3+: GNU GPL version  3  or
       later <http://gnu.org/licenses/gpl.html>.
       This  is  free  software: you are free to change and redistribute it.  There is NO WAR-
       RANTY, to the extent permitted by law.

SEE ALSO
       The full documentation for ls is maintained as a Texinfo manual.  If the  info  and  ls
       programs are properly installed at your site, the command

              info coreutils 'ls invocation'

       should give you access to the complete manual.

GNU coreutils 8.4               December 2011                          LS(1)
```

How to Locate a “man page”

```
$ man ls

$ man -k zip

$ man -M <myLocalManDir> <cmd name>
```

- The man command can search the NAME sections for keywords (-k option)
- The -M option can force man to read pages installed in non-default locations

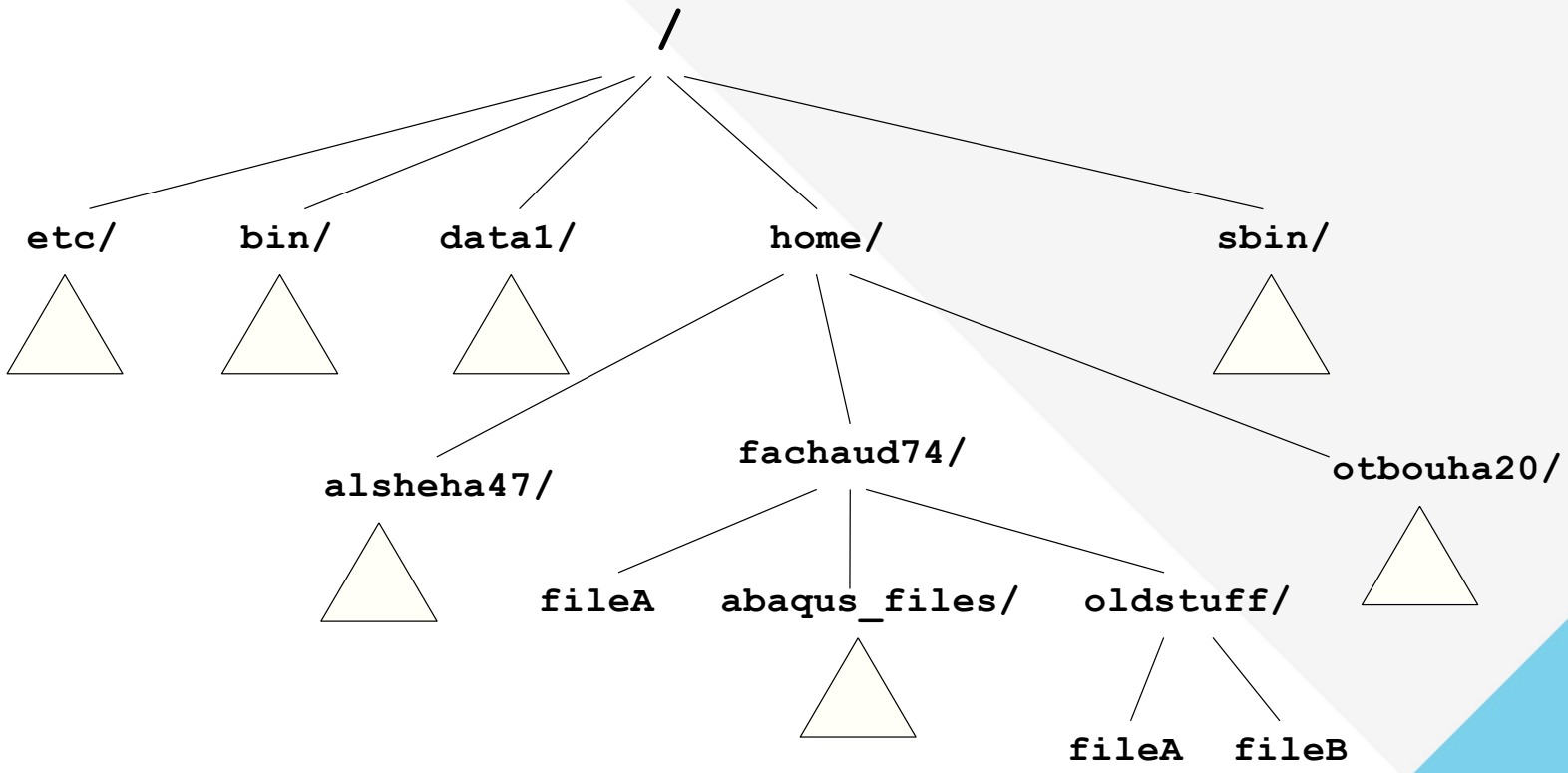
The Linux Filesystem

What is a Filesystem?

- Several definitions:
  - A directory structure contained within a disk drive or disk area
  - A method of organizing files on a disk.
  - A software mechanism that defines the way that files are named, stored, organized, and accessed on a storage medium.
  - A method for storing and organizing computer files and the data they contain to make it easy to find and access. File systems may use a storage device such as a hard disk or CD-ROM and typically involve maintaining the physical location of the files.

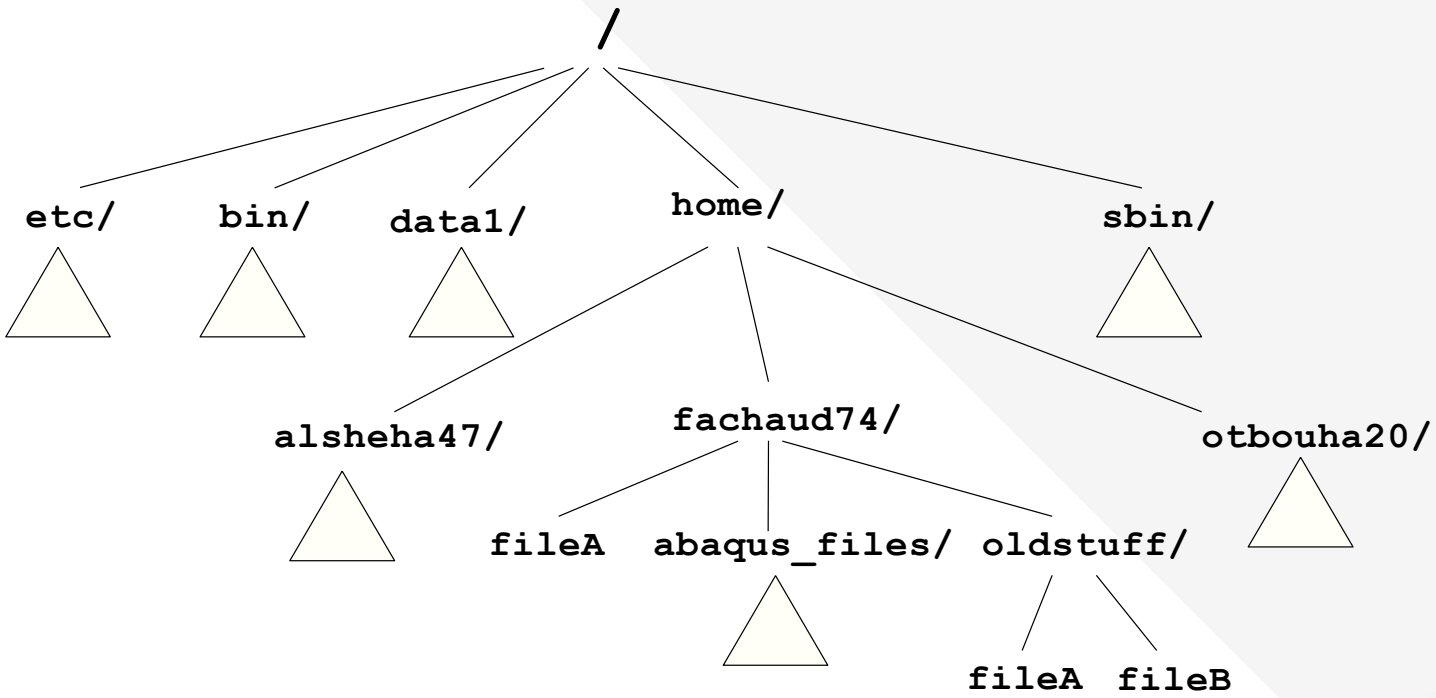
- Examples of filesystems: NTFS, FAT32, ext3, ext4, XFS, btrfs, JFS2, Luster, etc.

The Linux Filesystem Hierarchy



- The Linux filesystem consists of directories and sub-directories arranged in a tree structure

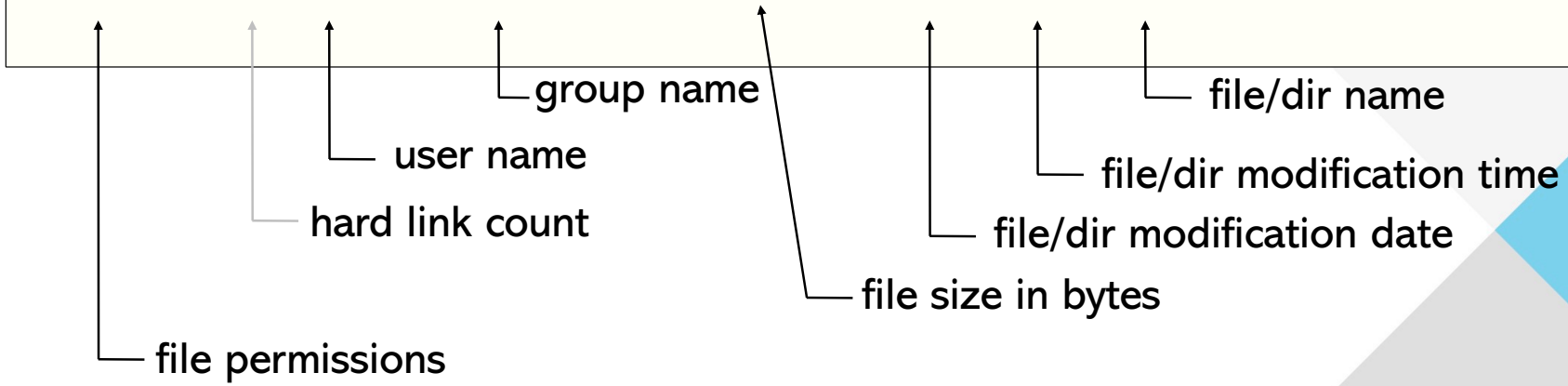
The Linux Filesystem Hierarchy



- Full pathname for `/home/fachaud74/oldstuff/fileA`
- The relative pathname for `fileB` , if our current working directory is `/home/fachaud74/`, is: `oldstuff/fileB`

Listing Directory Contents

```
[fachaud74@raad ~]$ ls -l
total 1976
drwxr-xr-x 12 fachaud74 rc.admin    4096 2009-01-29 11:28 bakexec_agent_files
drwxr-xr-x  2 fachaud74 rc.admin    4096 2003-09-02 13:00 bin
-rw-r--r--  1 fachaud74 rc.admin    1403 2009-03-09 11:23 cl1n044-bmc-eventlog.txt
-rw-------  1 fachaud74 rc.admin   15790 2009-03-22 13:54 suqoor-pbs-config.txt
drwxr-xr-x  2 fachaud74 rc.admin    4096 2009-03-12 08:29 Desktop
drwx------  3 fachaud74 rc.admin    4096 2009-02-04 14:03 Documents
-rw-r--r--  1 fachaud74 rc.admin 1310720 2009-02-23 12:02 file.db
-rw-r--r--  1 fachaud74 rc.admin    4776 2009-03-15 09:39 file.err
-rw-r--r--  1 fachaud74 rc.admin    6375 2009-03-15 09:40 file.log
-rw-r--r--  1 fachaud74 rc.admin       0 2009-03-15 09:41 file.out
```



Listing Directory Contents

```
$ ls [options] [directory or file name]
```

- Commonly used options

`-l`	display contents in “long” format
`-a`	show all files (including hidden files - those beginning with .)
`-t`	sort listing by modification time
`-r`	reverse sort order
`-F`	append type indicators with each entry (* / = @ |)
`-h`	print sizes in user-friendly format (e.g. 1K, 234M, 2G)

Changing Directories

```
$ cd [directory name]
```

- To switch to the most recent previously used directory:

```
$ cd -
```

- To switch to the parent directory of the current directory:

```
$ cd ..
```

Other Directory Commands

- To list the name of the present working directory:

```
$ pwd
```

- To make a new directory:

```
$ mkdir [directory name]
```

- To remove a directory (which must be empty):

```
$ rmdir [directory name]
```

Deleting Files

```
$ rm [options] [file name]
```

- Commonly used options
  - -i prompt user before any deletion
  - -r remove the contents of directories recursively
  - -f ignore non-existent files, never prompt

Better be fully awake before running this command!

```
$ rm -rf myWorkDirectory
```

Moving or Re-naming Files

```
$ mv [source] [target]
```

- If the source is a directory name, and…
  - target is an existing dir: source dir is moved inside target dir
  - target is new name: source dir is re-named to new name
- If the source is a file name, and…
  - target is an existing dir: source file is moved inside target dir
  - target is a new name: source file is re-named to new name

Copying Files

```
$ cp [options] [source] [target]
```

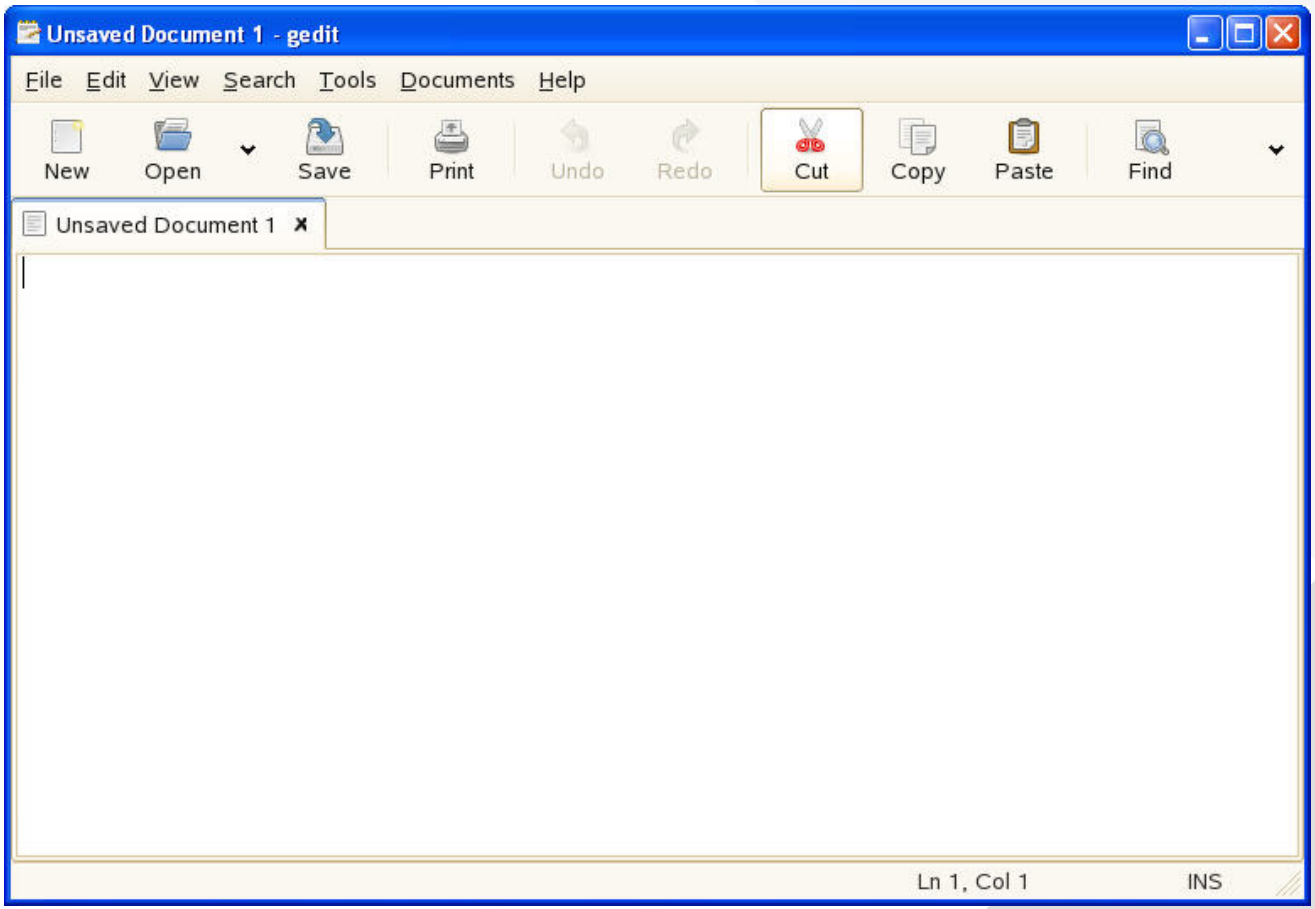
- If source is a file, and…
  - target is a new name: duplicate source and call it target
  - target is a directory: duplicate source, with same name, and place it in directory
- If source is a directory and the -R option is used…
  - Target is a new name: copy directory and its contents recursively into directory with new name
  - Target is a directory: duplicate source, with same name, and place it in directory

Exercises

- How can you determine your present working directory?
- Change you directory so you are in the `/proc` directory.
- List the files in that location.
- How would you go back to your home directory?
- In your home directory, create a new sub-directory called "myNewDir".
- Move into this sub-directory
- How would you delete the myNewDir directory?

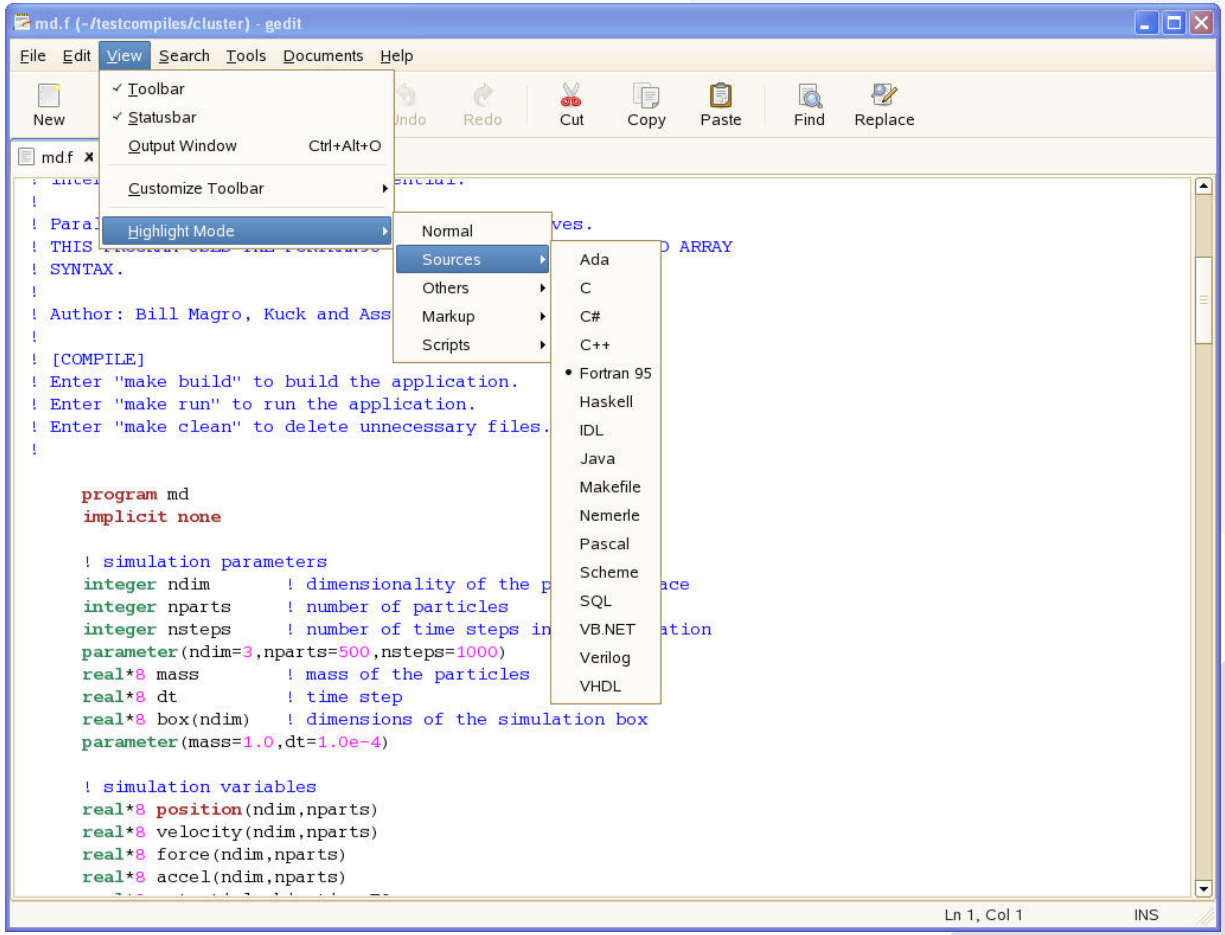
How to Edit an ASCII File

- Use the gedit text editor.
- More advanced editors include vi and emacs.



How to Edit an ASCII File

- The gedit program has extensive syntax-highlighting features



File Ownership and Permissions

```
-rwx--x--x  1 fachaud74 rc.admin     826 2009-03-15 09:40 script.pl
```

permissions

user and group ownership

Decimal	Binary	Permissions
0	000	---
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx

- There are 3 sets of permissions for each file
  - 1st set - **u**ser (who owns the file)
  - 2nd set - **g**roup (to which file owner belongs)
  - 3rd set - **o**ther (all other users)
- The r indicates read permission
- The w indicates write permission
- The x indicates execute permission

Directory Permissions

```
drwxr-xr-x  2 fachaud74 rc.admin    4096 2007-06-25 18:48 public_html
```

- The meanings of the permission bits for a directory are slightly different than for regular files:
  - `-r` permission means the user can list the directory's contents
  - `-w` permission means the user can add or delete files from the directory
  - `-x` permission means the user can cd into the directory; it also means the user can execute programs stored in it
- Notice that if the file is a directory, the leading bit before the permissions is set to `d`, indicating directory.

The ‘chmod’ Command

```
$ chmod [options] [permission mode] [target]
```

```
$ chmod 777 myFile.txt      ( the permissions will be set to rwxrwxrwx )

$ chmod o-x myFile.txt      ( the permissions will change to rwxrwxrw- )

$ chmod gu-x myFile.txt     ( the permissions will change to rw-rw-rw- )

$ chmod u+x myFile.txt      ( the permissions will change to rwxrw-rw- )
```

-R is a commonly used option. It recursively applies the specified permissions to all files and directories within target, if target is a directory.

Exercises

- Create a new directory called “myNewDir”
- In this directory, create a new file called “newfile.txt” using gedit (write “hello neighbor!” in the file).
- Change the permissions on this file; give it all possible permissions for all three tuples
- Can any of your neighbors view the contents of this file with the less command?
- Which command would one use to change the group ownership of a file? (not covered in slides)

Viewing File Contents with ‘less’

```
$ less [filename(s)]
```

This is a flexible utility to view the contents of files, allowing for the ability, among others, to scroll backwards as well as forwards.

Here are some commonly used keystrokes, employed while content is being displayed on screen:

d	scroll down 1 page
b	scroll back 1 page
/*PATTERN*	find the next occurrence of *PATTERN* in the file
?*PATTERN*	find previous occurrence of *PATTERN* in the file
n	repeat the previous search

>	go to the end of the file
<	go to beginning of the file
*N***g**	go to line *N* of the file
*N***p**	go to a position *N*% into file
m*C*	mark current position with the lower case character *C*
'*C*	return to position previously marked with *C*

File Contents with ‘head’ & ‘tail’

```
$ head [-n [-]N] [filename]
```

```
$ tail [-n N | -f] [filename]
```

These commands display the leading or trailing portions of a file, respectively.

head *filename*
prints the first 10 lines of the file

head –n *N filename*
prints the first *N* lines of the file

head –n -*N filename*
prints all but the last *N* lines of file

tail *filename*
prints the last 10 lines of the file

tail –n *N filename*
prints the last *N* lines of the file

tail –f *filename*
prints last 10 lines, then keeps appending data as the file grows

Searching for Files with 'find'

```
$ find [target dir] [expression]
```

```
$ find . -name "*.txt" -print
```
(search current dir, print all files with names ending in suffix .txt)

```
$ find . -newer results4.dat -name "*.dat" -print
```
(search current dir, print files newer than results4.dat with names ending in suffix .dat)

```
$ find /scratch/user1 -mtime +2  -print
```
(search user1's scratch dir, print names of files modified more than 2 days ago)

```
$ find /scratch/user1 -mtime -7  -print
```
(search user1's scratch dir, print names of files modified within last week)

```
$ find /tmp -user user1 -print
```
(search tmp dir, print names of all files belonging to user user1)

```
$ find /scratch/user1 -name core -exec rm '{}' \;
```
(search user1's scratch dir, delete any files named 'core')

The ‘egrep’ Command

```
$ egrep [options] PATTERN [file or dir name]
```

- Used to search the specified file(s) for lines containing a match to the given *PATTERN (an extended regular expression)*.

- Commonly used options:
  - **-A** *NUM* print *NUM* lines of trailing context after matching lines
  - **-B** *NUM* print *NUM* lines of leading context before matching lines
  - **-i** ignore case distinctions in both *PATTERN* and input files
  - **-n** prefix each line of output with the line # within the input files
  - **-R** read all files under each directory, recursively
  - **-v** reverse the sense of matching; list *non*-matching lines
  - **-H** print the file name for each match

The ‘egrep’ Command

As an example, we examine the contents of some text file called “part-info.txt”. For a better understanding of what egrep will do, here is what the beginning of this file looks like:

```
PartitionName=s_short
   AllowGroups=ALL AllowAccounts=ALL AllowQos=ss,scustom1,scustom2,scustom3
   AllocNodes=ALL Default=NO QoS=N/A
   DefaultTime=02:00:00 DisableRootJobs=NO ExclusiveUser=NO GraceTime=0 Hidden=NO
   MaxNodes=UNLIMITED MaxTime=08:00:00 MinNodes=1 LLN=NO MaxCPUsPerNode=UNLIMITED
   Nodes=nid00[200-225]
   PriorityJobFactor=1 PriorityTier=1 RootOnly=NO ReqResv=NO OverSubscribe=YES:4
   OverTimeLimit=NONE PreemptMode=OFF
   State=UP TotalCPUs=1248 TotalNodes=26 SelectTypeParameters=NONE
   DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED

PartitionName=s_long
   AllowGroups=ALL AllowAccounts=ALL AllowQos=sl,scustom1,scustom2,scustom3
   AllocNodes=ALL Default=NO QoS=N/A
   DefaultTime=1-00:00:00 DisableRootJobs=NO ExclusiveUser=NO GraceTime=0 Hidden=NO
   MaxNodes=UNLIMITED MaxTime=7-00:00:00 MinNodes=1 LLN=NO MaxCPUsPerNode=UNLIMITED
   Nodes=nid00[200-225]
   PriorityJobFactor=1 PriorityTier=1 RootOnly=NO ReqResv=NO OverSubscribe=YES:4
   OverTimeLimit=NONE PreemptMode=OFF
   State=UP TotalCPUs=1248 TotalNodes=26 SelectTypeParameters=NONE
   DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED

PartitionName=s_debug
   AllowGroups=ALL AllowAccounts=ALL AllowQos=sd,scustom1,scustom2,scustom3
            .
            .
            .
```

The ‘egrep’ Command

Now, we will print only the lines that list the name of the respective partition at the beginning of a line, and also the lines that list the state of that partition.

```
$ egrep "^P.*|State=" part-info.txt
PartitionName=s_short
   State=UP TotalCPUs=1248 TotalNodes=26 SelectTypeParameters=NONE
PartitionName=s_long
   State=UP TotalCPUs=1248 TotalNodes=26 SelectTypeParameters=NONE
PartitionName=s_debug
   State=UP TotalCPUs=96 TotalNodes=2 SelectTypeParameters=NONE
PartitionName=l_short
   State=UP TotalCPUs=6528 TotalNodes=136 SelectTypeParameters=NONE
PartitionName=l_long
   State=UP TotalCPUs=6528 TotalNodes=136 SelectTypeParameters=NONE
PartitionName=express
   State=UP TotalCPUs=192 TotalNodes=4 SelectTypeParameters=NONE
$
```

Extended Regular Expressions

- A regular expression (RE) is a pattern that describes a set of strings. As with arithmetic expressions, REs are constructed by using various operators to combine smaller expressions.
- The fundamental building blocks of a RE are REs that match a single character:

`A ... Z`	the characters 'A' through 'Z' match themselves
`a ... z`	the characters 'a' through 'z' match themselves
`0 ... 9`	the digits '0' through '9' match themselves
`[abc]`	this is a *bracket expression* that matches a single char: either 'a' or 'b' or 'c'
`[a-z]`	bracket expressions may contain ranges; this one matches any single char between 'a' and 'z'; **`[a-zA-Z0-9]`** would match any alphanumeric char
`[^xyz]`	matches any single char other than 'x' or 'y' or 'z'
`.`	(a dot) matches any single character

Extended Regular Expressions

- An RE may be followed by one of several repetition operators:

  `?` This means the preceding item is optional and matched at most once

  `*` The preceding item will be matched 0 or more times

  `+` The preceding item will be matched 1 or more times

- REs may be concatenated (logical and) or alternated (logical or, using the '`|`' operator):

  `EXPR1EXPR2` matches any string formed by two substrings that respectively match the concatenated sub-expressions EXPR1 and EXPR2

  `EXPR1|EXPR2` matches either sub-expression EXPR1 or EXPR2

- There are also some characters with special meanings (meta-characters):

  `^` (used outside of a bracket expr) matches the empty string at the beginning of a line

  `$` matches the end of a line

Archiving Files with ‘tar’

```
$ tar [options] [archive file] [file or dir name]
```

- Used to “package” multiple files (along with directories if any) into one archive file (suffixed with `.tar` by convention).

- Commonly used options
  - `x` extract files from an archive
  - `c` create a new archive
  - `t` list the contents of an archive
  - `v` verbosely list files processed
  - `f` use the specified archive file

‘tar’ Examples

```
$ tar -cvf myHomeDir.tar .
```
(archive the current dir into a file called myHomeDir.tar)
```
$ tar -tvf myHomeDir.tar
```
(show the TOC for the named archive)
```
$ tar -xvf myHomeDir.tar ./abaqus_files
```
(extract the dir abaqus_files from archive)

- Be careful when extracting files.
- Where files are extracted depends on how they were packaged.
- Always a good idea to check TOC (-t option) before extraction.

Input/Output (I/O) Re-direction

Input and Output (I/O)

- Every process needs to communicate with a number of outside entities in order to do useful work
  - It may require input to work with. This input may come from the keyboard, from a file stored on disk, from a joystick, etc.
  - It may produce output resulting from its work. This data will need to be sent to the screen, written to a file, sent to the printer, etc.
- In unix, anything that can be read from or written to can be treated like a file (terminal display, file, keyboard, printer, memory, network connection, etc.)
- Each process references such “files” using small unsigned integers (starting from 0) stored in its file descriptor table.
- These integers, called file descriptors, are essentially pointers to sources of input or destinations for output.

Redirection Operators

`<`	redirects input
`>`	redirects output
`>>`	appends output
`<<`	input from *here document*
`2>`	redirects error
`&>`	redirects output and error
`>&`	redirects output and error
`2>&1`	redirects error to where output is going
`1>&2`	redirects output to where error is going

Redirection Operators

```
$ ls
dir1 dir2 file1
$ ls > file_list.txt
$ cat file_list.txt
dir1
dir2
file1
$ ls >> file_list.txt
$ sort < file_list.txt
dir1
dir1
dir2
dir2
file1
file1
$ sort < file_list.txt > sorted_file_list.txt
$ cat sorted_file_list.txt
dir1
dir1
dir2
dir2
file1
file1
$ cc buggy.c 2> errfile
```

(save any compilation errors in file errfile)

```
$ find . -name "*.c" -print > foundit 2> /dev/null
```

(Find C source file names and save in file "foundit", throwing away errors)

Pipes

- A pipe takes the output of the command to the left of the pipe symbol (|) and sends it to the input of the command listed to its right.
- A 'pipeline' can consist of more than one pipe.

```
$ who > tmp
$ wc -l tmp
38 tmp
$ rm tmp
```

(using a pipe saves disk space and time)

```
$ who | wc -l
38
```

Pipes: An Example

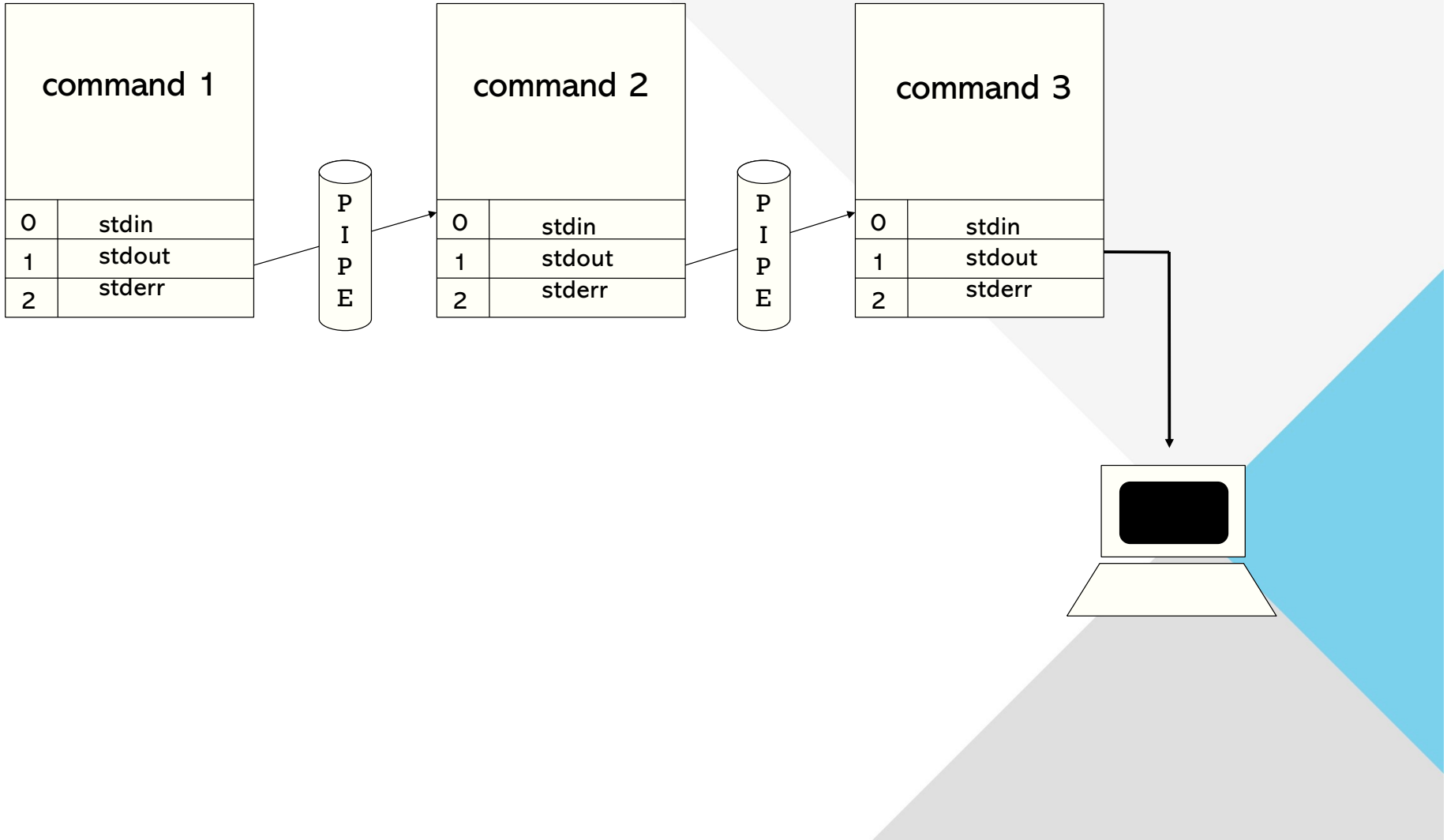
```
$ curl -s "https://en.wikipedia.org/wiki/Linux" | sed 's/[^a-zA-Z ]/ /g' | \
> tr 'A-Z ' 'a-z\n' | egrep '[a-z]' | sort -u | less

a
ability
able
about
aboutsite
abuse
abusing
accept
accesskey
account
across
action
actions
active
ad
.
.
.
```

example courtesy of: http://en.wikipedia.org/wiki/Pipeline_(Unix)

1. **curl** obtains the HTML contents of a web page.
2. **sed** removes all non-alphabetic characters from the content, replacing them with spaces.
3. **tr** changes all of the uppercase letters into lowercase and converts the spaces in the lines of text to newlines (each 'word' is now on a separate line).
4. **egrep** includes only lines that contain at least one lowercase alphabetical character (removing any blank lines).
5. **sort** sorts the list of 'words' into alphabetical order, and the -u switch removes duplicates.
6. **less** allows the user to scroll through the results.

Pipes


command 1
0 stdin
1 stdout
2 stderr
PIPE
command 2
0 stdin
1 stdout
2 stderr
PIPE
command 3
0 stdin
1 stdout
2 stderr

Exercises

- In your home directory, type: `man 1 intro > test.txt`
- View the file contents with the `less` command
- Look at the “head” of the file, as well as the “tail” with the corresponding commands
- Print the lines (from this file) containing the digit “6”
- Count the number of lines containing “6” (use the `wc` command)
- What is the total number of lines in the file?

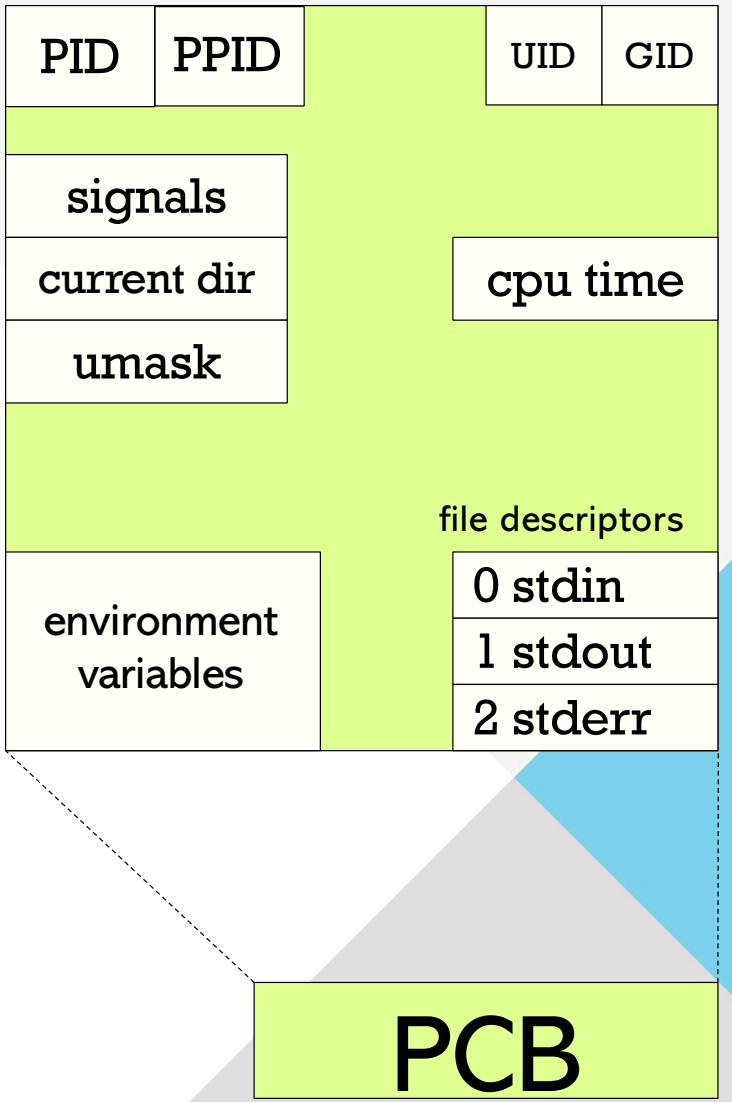
UNIX Processes

What is a Process?

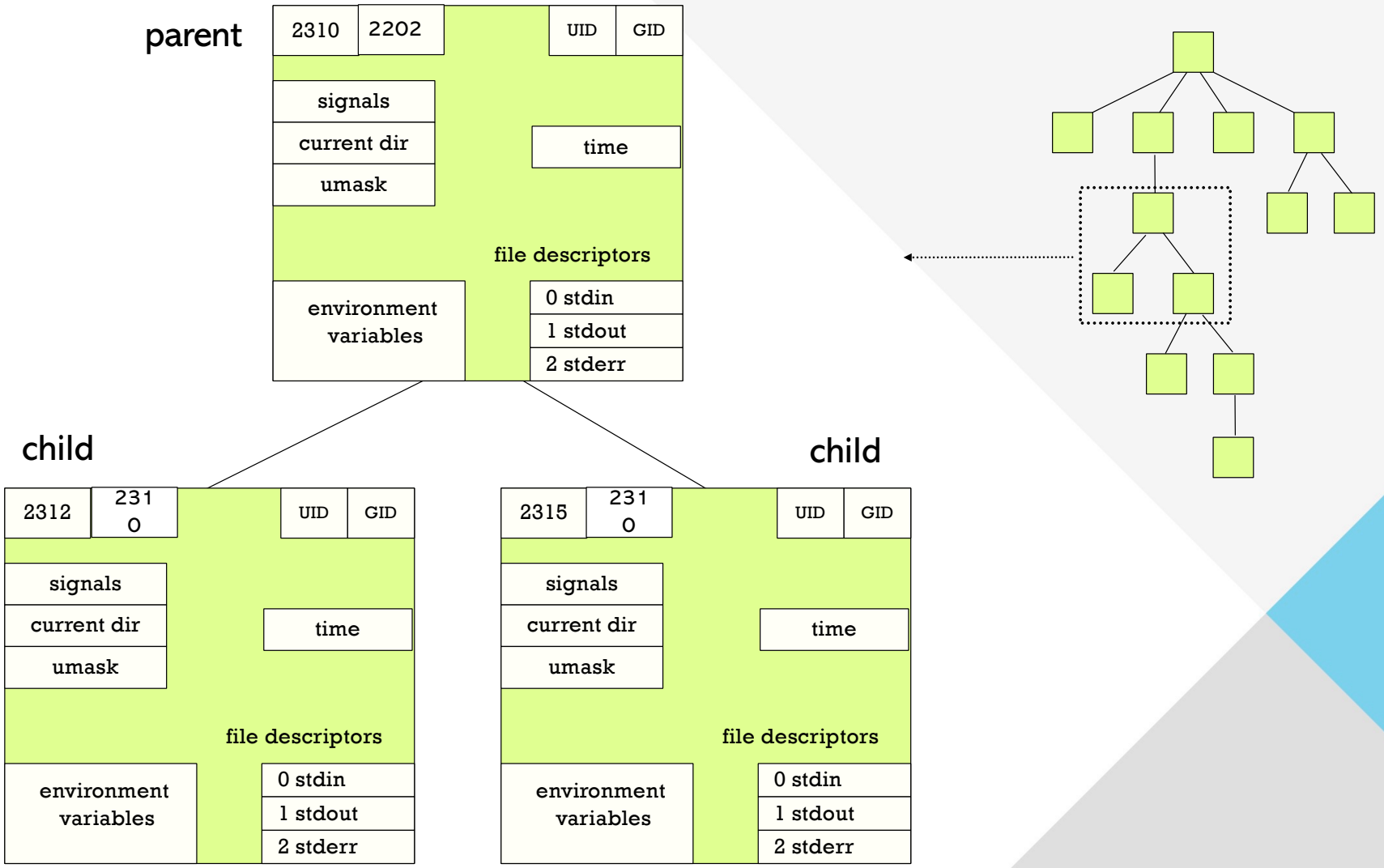
- Program
  - Machine readable code (binary) that is stored on disk
- Process
  - A program that is loaded into memory and executed
- The operating system ("kernel") controls and manages processes
  - It allows multiple processes to share the CPU (multi-tasking)
  - Manages resources (e.g. memory, I/O)
  - Assigns priorities to competing processes
  - Facilitates communication between processes
  - Can terminate (kill) processes

Components of a Process

- **PID:** each process is uniquely identified by an integer (process ID), assigned by the kernel
- **PPID:** the PID of the parent process
- **UID:** integer ID of the user to whom the process belongs
- **GID:** integer ID of the user group to which the process belongs
- **Signals:** how this process is configured to respond to various signals
- **Current dir:** the current working directory of the process
- **Environment variables:** variables that customize the behaviour of the process (e.g. PATH)
- **File descriptors:** a table of small unsigned integers, starting from 0, that reference open "files" used by the process (for receiving input or producing output).



The Process Hierarchy



The Process Hierarchy

- Processes are associated in parent-child relationships
  - A process can create or “spawn” another process and therefore become the “parent” of the created process. The spawned process becomes the “child”.
  - A process can have multiple children, but every child can only have one parent.
  - The “family tree” of processes on the system constitutes the process hierarchy.
  - A child process inherits various characteristics from its parent at creation time.

The ‘ps’ Command

- The `ps` command gives a snapshot of all processes running on the system at any given moment.
- It tells us who is running processes on the system as well as how much time the system is spending on each process, and other process characteristics.
- `ps` has many versions and has an extensive set of options. We will illustrate the version installed on suqoor, with some commonly used options.

```
$ ps [options]
```

- Commonly used options (ps can take different styles of options)

a	select all processes on a terminal (including those of other users)
x	select processes without controlling terminals
u	associate processes w/ users in the output
w	display output in wide column format
-U *username*	select processes belonging to specific user
-j	display in jobs format (info about groups of related procs)
-e	select all processes
-l	display in long format
-f	display in full format

The ‘ps’ Command

```
$ ps a
  PID TTY      STAT   TIME COMMAND
 8250 tty7     Ss+    2:32 /usr/X11R6/bin/X :0 -audit 0 -br -auth /var/lib/gdm/:0.Xauth -nolisten tcp vt7
 8513 tty2     Ss+    0:00 /sbin/mingetty tty2
 8514 tty3     Ss+    0:00 /sbin/mingetty tty3
 8515 tty4     Ss+    0:00 /sbin/mingetty tty4
 8516 tty5     Ss+    0:00 /sbin/mingetty tty5
 8517 tty6     Ss+    0:00 /sbin/mingetty tty6
12784 pts/1    Ss+    0:00 bash
29732 pts/56   Ss+    0:00 -bash
31089 pts/56   S      0:02 emacs README
 6410 pts/53   Ss+    0:00 -bash
12786 pts/54   Ss+    0:00 -bash
14731 pts/55   Ss     0:00 bash
14774 pts/55   R+     0:00 ps a
$ ps au
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root      8250  0.0  0.0  32980  9248 tty7     Ss+  Mar16   2:32 /usr/X11R6/bin/X :0 -audit 0 -br -auth
/var/lib/gdm/:0.Xauth -nolisten tcp vt7
root      8513  0.0  0.0   3064   692 tty2     Ss+  Mar16   0:00 /sbin/mingetty tty2
root      8514  0.0  0.0   3064   692 tty3     Ss+  Mar16   0:00 /sbin/mingetty tty3
root      8515  0.0  0.0   3060   688 tty4     Ss+  Mar16   0:00 /sbin/mingetty tty4
root      8516  0.0  0.0   3060   692 tty5     Ss+  Mar16   0:00 /sbin/mingetty tty5
root      8517  0.0  0.0   3064   692 tty6     Ss+  Mar16   0:00 /sbin/mingetty tty6
7006     12784  0.0  0.0  10116  2404 pts/1    Ss+  Mar17   0:00 bash
7006     29732  0.0  0.0   9212  2340 pts/56   Ss+  Mar17   0:00 -bash
7006     31089  0.0  0.0  41248 12376 pts/56   S    Mar17   0:02 emacs README
7030      6410  0.0  0.0  10112  2404 pts/53   Ss+  07:35   0:00 -bash
7030     12786  0.0  0.0  10112  2444 pts/54   Ss+  08:29   0:00 -bash
7001     14731  0.0  0.0  10116  2384 pts/55   Ss   08:45   0:00 bash
7001     14779  0.0  0.0   3500   900 pts/55   R+   08:48   0:00 ps au
```

The ‘ps’ Command

```
$ ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.0    780   304 ?        S    Mar16   0:02 init [5]
root          2  0.0  0.0      0     0 ?        S    Mar16   0:00 [migration/0]
root          3  0.0  0.0      0     0 ?        SN   Mar16   0:00 [ksoftirqd/0]
root          4  0.0  0.0      0     0 ?        S    Mar16   0:00 [migration/1]
root          5  0.0  0.0      0     0 ?        SN   Mar16   0:00 [ksoftirqd/1]
root          6  0.0  0.0      0     0 ?        S    Mar16   0:00 [migration/2]
root          7  0.0  0.0      0     0 ?        SN   Mar16   0:00 [ksoftirqd/2]
root          8  0.0  0.0      0     0 ?        S    Mar16   0:00 [migration/3]
.
.
.
postfix   10095  0.0  0.0  21296  2084 ?        S    08:07   0:00 pickup -l -t fifo -u
root      12779  0.0  0.0  39452  2716 ?        Ss   08:29   0:00 sshd: jomalek58 [priv]
7030      12782  0.0  0.0  39452  1860 ?        S    08:29   0:00 sshd: jomalek58@pts/54
7030      12786  0.0  0.0  10112  2444 pts/54   Ss+  08:29   0:00 -bash
root      14695  0.0  0.0  39456  2664 ?        Ss   08:45   0:00 sshd: fachaud74 [priv]
7001      14698  0.0  0.0  39752  1936 ?        S    08:45   0:00 sshd: fachaud74@notty
7001      14699  0.0  0.0  35976  6952 ?        Rs   08:45   0:00 /usr/bin/xterm
7001      14731  0.0  0.0  10116  2384 pts/55   Ss   08:45   0:00 bash
7001      15345  0.0  0.0   3500   900 pts/55   R+   08:51   0:00 ps aux
```

The ‘ps’ Command

```
$ ps u -U fachaud74
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
7001     11421  0.0  0.0 114500  1212 pts/0    R+   14:43   0:00 ps u -U fachaud74
7001     13115  0.0  0.0 100220  2432 ?        S    Nov27   0:04 sshd: fachaud74@notty
7001     21533  0.0  0.0 100172  2288 ?        S    Dec02   0:01 sshd: fachaud74@notty
7001     21534  0.0  0.0 161068  7120 ?        Ss   Dec02   0:00 /usr/bin/xterm -ls
7001     21552  0.0  0.0 110564  1956 pts/0    Ss   Dec02   0:00 -bash
7001     30173  0.0  0.0  97856  1864 ?        S    Dec02   0:00 sshd: fachaud74@pts/2
7001     30174  0.0  0.0 110564  1952 pts/2    Ss+  Dec02   0:00 -bash
$
```

Killing Processes with 'kill'

```
$ kill -l
$ kill [signal name] pid
```

- The **`kill -l`** command lists all the signal names available.
- The **`kill`** command can generate a signal of any type to be sent to the process specified by a PID.
- The **`kill -9`** sends the (un-interruptable) kill signal.
- '**`kill`**' is the not the best name for this command, because it can actually generate any type of signal.

The Bash Shell

What is a Shell?

- A shell is the program that interprets what the user types at the command line, and executes the user's commands.
- When the shell program is in execution, it is considered, of course, a process.
- Historically, the developer community has contributed to the development of various shell programs.
- The various shells provide the same core functionality but also differ to varying degrees in other features they provide.
- Some shells are better suited to scripting or shell programming (covered later)

Example of What a Shell Does

```
$ cat *.txt | wc > txt_files_size.$USER
$ echo "Date? `date`" >> txt_files_size.$USER
```

1) command line is broken into "words" (or "tokens")
2) quotes are processed
3) redirection and pipes are set up
4) variable substitution takes place
5) command substitution takes place
6) filename substitution (globbing) is performed
7) command/program execution

The Exit Status of a Process

- After a command or program terminates, it returns an "exit status" to the parent process.
- The exit status is a number between 0 and 255.
  - By convention, exit status 0 means successful execution
  - Non-zero status means the command failed in some way
  - If the command was not found by shell, status is 127
  - If the command dies due to a fatal signal, status is 128 + sig #
- After command execution, type **`echo $?`** at command line to see its exit status number.

```
$ egrep root /etc/passwd
root:x:0:0:root:/root:/bin/bash
$ echo $?
0
$ egrep barackobama /etc/passwd
$ echo $?
1
$
```

Multiple Commands and Grouping

- A single command line can consist of multiple commands. Each command must be separated from the previous by a semicolon. The exit status returned is that of the last command in the chain.

```
$ ls; pwd; date
```

- Commands can also be grouped so that all of the output is either piped to another command or redirected to some file.

```
$ ( ls; pwd; date ) > outputfile
```

Conditional Execution and Backgrounding

- Two command strings can be separated by the special characters && or || . The command on the right of either of these metacharacters will or will not be executed based on the exit status of the command on the left.

```
$ cc program1.c -o program1.exe && program1.exe
$ cc program2.c -o program2.exe >& err || mail joe.shmoe@qatar.tamu.edu < err
```

- By placing an & at the end of a command line, the user can force the command to run in the "background". The user will not have to wait for the command to finish before receiving the next prompt from the shell.

```
$ program1.exe > p1_output &
[1] 1557
$
```

Command Line Shortcuts

- Command and filename completion
  - A feature of the shell that allows the user to type partial file or command names and completes the rest itself
- Command history
  - A feature that allows the user to “scroll” through previously typed commands using various key strokes
- Aliases
  - A feature that allows the user to assign a simple name even to a complex combination of commands
- Filename substitution
  - Allows user to use “wildcard” characters (and other special characters) within file names to concisely refer to multiple files with simple expressions

Command and Filename Completion

- To save typing, bash provides a mechanism that allows the user to type part of a command name or file name, press the tab key, and the rest of the word will be completed for the user.

```
$ ls
file1 file2 foo foobarckle fumble
$ ls fu[tab]
$ ls fumble              (expands fu to fumble)
$ ls fx[tab]                        (terminal beeps, nothing happens)
$ ls fi[tab]
$ ls file                (expands fi to file)
$ ls fi[tab][tab]
file1 file2
$ ls fi                              (lists all possibilities)
$ da[tab]
$ date                               (expands to the date command)
```

Command History

- The history mechanism keeps a numbered record of commands entered by the user at the command line.
  - during a login session, this list is stored in memory
  - upon exit, list is appended to a history file (`~/.bash_history`)
- The up and down arrow keys can be used to "scroll" through the history list at the command line.
- The **`history`** built-in command can display the history of commands (preceded by an event number).
- The **`!`** character can also be used to re-execute old commands.

The ‘history’ Command

```
$ history

   .
   .
   .

  993  qstat -r
  994  p_qstat 11324
  995  kill -l
  996  man 7 signal
  997  man 7 signal
  998  stty
  999  su -
 1000  su -
 1001  history
$
```

Filename Substitution

- Meta-characters are special characters used to represent something other than themselves.
- When evaluating the command line the shell uses meta-characters to abbreviate filenames or pathnames that match a certain set of characters.
- The process of expanding meta-characters into filenames is called filename substitution, or globbing.
- Meta-characters used for filename substitution:

`*`	matches 0 or more characters
`?`	matches exactly 1 character
`[abc]`	matches 1 character in the set: a, b, or c
`[!abc]` b, or c	matches 1 char not in the set: anything other than a,
`[!a-z]`	matches 1 character not in the range from a to z
`{a,ile,ax}`	matches for a character or a set of characters

Filename Substitution

```
$ ls *
abc abc1 abc122 abc123 abc2 file1 file1.bak file2 file2.bak none
nonsense nobody nothing nowhere one
$ ls *.bak
file1.bak file2.bak
$ echo a*
abc abc1 abc122 abc123 abc2
$ ls a?c?
abc1 abc2
$ ls ??
ls: ??: No such file or directory
$ echo abc???
abc122 abc123
$ echo ??
??
$ ls abc[123]
abc1 abc2
$ ls abc[1-3]
abc1 abc2
$ ls [a-z][a-z][a-z]
abc one
```

Filename Substitution

```
$ ls *
a.c b.c abc ab3 ab4 ab5 file1 file2 file3 file4 file5 foo faa fumble
$ ls f{oo,aa,umble}
foo faa fumble
$ ls a{.c,c,b[3-5]}
a.c ab3 ab4 ab5
$ mkdir /home/student01/mycode/{old,new,dist,bugs}
$ echo fo{o, um}*
fo{o, um}*
```

- To use a meta-character as a literal character, use the backslash to prevent the meta-character from being interpreted.

```
$ ls
abc file1 youx
$ echo How are you?
How are youx
$ echo How are you\?
How are you?
$ echo When does this line \
>ever end\?
When does this line ever end?
```

Filename Substitution

- The tilde character (~) by itself evaluates to the full pathname of the user's home directory.
- When prepended to a username, the tilde expands to the full pathname of that user's home directory.
- When the plus sign follows the tilde, the value of the present working directory is produced.
- When the minus sign follows, the value of the previous working directory is produced.

```
$ echo ~
/ddn/home/fachaud74
$ echo ~otbouha20
/ddn/home/otbouha20
$ echo ~+
/ddn/home/fachaud74
$ echo ~-
/ddn/home/fachaud74/mycode
```

Variables

What Are Variables?

- They are names that store pieces of text for later use
- They help you avoid re-typing long paths, filenames, or settings
- Think of them like labelled sticky notes that the shell keeps for you while you work

```
$ cd /very/long/path/to/my/project/directory
$ ls /very/long/path/to/my/project/directory
$ cp file.txt /very/long/path/to/my/project/directory
```

```
$ PROJECT=/very/long/path/to/my/project/directory
$ cd $PROJECT
$ ls $PROJECT
$ cp file.txt $PROJECT
```

Variable Scope & Naming

- There are two types of variables: local and environment
  - Local: known only to the shell in which they are created
  - Environment: available to any child processes spawned from the shell in which they were created
  - Some variables are user-created, other are special shell variables (e.g. PWD)
- Variable names must begin with an alphabetic or underscore (_) character. The remaining characters can be alphabetic, numeric, or the underscore (any other characters mark the end of the variable name).
  - Names are case-sensitive
  - When assigning values, no whitespace must surround the = sign
  - To assign a null value, follow the = sign with a newline

Creating Variables

- The simplest way of creating a local variable:

```
$ myname=Faisal                    (simplest format: variable=value)
```

- The declare built-in command can also be used to create variables.
  - Without arguments, declare lists all currently set variables
  - It is possible to create read-only vars; these cannot be unset, but they can be reassigned values

```
$ declare myname=Faisal
```

- The value stored in any variable can be extracted by pre-pending the $ sign to the variable name.

Some Special Variables

- Not all vars are created by the user. Some are created and updated by the shell.

$$	The PID of the current shell process
$?	The exit value of the last executed command
$!	The PID of the last job put in the background

```
$ echo $$
6125
$ ls
dir1  dir2  file.txt
$ cd dir37
-bash: cd: dir37: No such file or directory
$ echo $?
1
$ echo $!

$ sleep 60 &
[1] 29843
$ echo $!
29843
```

The Scope of Local Variables

- Only visible within the shell in which they are created.

```
$ echo $$
1313
$ round=world
$ echo $round
world
$ bash                  (start a subshell)
$ echo $$
1326
$ echo $round

$ exit                  (exit subshell, return to parent shell)
$ echo $$
1313
$ echo $round
world
```

Environment Variables

- Environment variables are variables that have been declared using **`declare -x`** or have been "exported" using the **`export`** built-in command.

```
$ MYNAME=Faisal
$ export MYNAME
$ export NAME="Faisal Chaudhry"
$ declare -x HISNAME="Faisal Chaudhry"
```

- Their scope is not limited to the current shell. They are available to any child process of the shell in which they are created.
- They are also referred to as "global variables".
- By convention, environment variables are capitalized.
- Some environment variables, such as HOME, LOGNAME, PATH, and SHELL are set by the system as the user logs in.
- Some environment variables are often defined in various shell start-up (initialization) files.
- The **`export -p`** command lists all environment (or global) variables currently defined.

Bash Environment Variables

GROUPS	an array of group IDs to which current user belongs
HISTSIZE	# of commands to remember in command history
HOME	pathname of current user's home directory
PATH	the search path for commands. It is a colon separated list of directories in which the shell looks for commands.
PPID	PID of the parent process of the current shell
PS1	primary prompt string ('$' by default)
PS2	secondary prompt string ('>' by default)
PWD	present working directory (set by cd)
SHELL	the pathname of the current shell
USER	the username of the current user

The Search Path

- The shell uses the PATH environment variable to locate commands typed at the command line
- The value of PATH is a colon separated list of full directory names.
- The PATH is searched from left to right. If the command is not found in any of the listed directories, the shell returns an error message
- If multiple commands with the same name exist in more than one location, the first instance found according to the PATH variable will be executed.

```
$ echo $PATH
/lustre/home/fachaud74/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games:/
usr/lib/mit/bin:/usr/lib/mit/sbin:/opt/cray/pe/bin:/lustre/sw/xc40ac/local/script
s/slurm
```

Quoting

Quoting

- Quoting is used to protect special metacharacters from interpretation and prevent parameter expansion
- There are 3 methods of quoting
  - the backslash
  - single quotes
  - double quotes
- Single and double quotes must be matched
- Single quotes protect metacharacters like $, *, ?, | , > and < from interpretation
- Double quotes do the same but allow variable and command substitution to take place

The Backslash

- The backslash (\ ) is used to quote (or “escape”) a single character from interpretation
- \ is not interpreted if placed in single quotes.
- \ will protect the $, backquotes (` `), and the backslash from interpretation if enclosed in double quotes.

```
$ echo Where are you going\?
Where are you going?
$ echo \\
\
$ echo '\\'
\\
$ echo '\$5.00'
\$5.00
$ echo "\$5.00"
$5.00
$ echo 'Don\'t you need $5.00?'
>
> '
Don\t you need .00?
```

Single Quotes

- Single quotes protect all metacharacters from interpretation. To print a single quote, it must be enclosed in double quotes or escaped with a backslash.

```
$ echo 'hi there
> how are you?
> when will this end?
> when the quote is matched
> oh'
hi there
how are you?
when will this end?
when the quote is matched
oh
$ echo Don\'t you need '$5.00?'
Don't you need $5.00?
$ echo 'Mother yelled, "Time to eat!"'
Mother yelled, "Time to eat!"
```

Double Quotes

- Double quotes allow variable and command substitution, and protect any other metacharacters from interpretation by the shell.

```
$ name=Faisal
$ echo ”Hi $name, I’m glad to meet you!”
Hi Faisal, I’m glad to meet you!
$ echo ”Hey $name, the time is $(date)”
Hey Faisal, the time is Sun Sep 12 8:30:34 AST 2014
```

Command Substitution

- Used when:
  - assigning the output of a command to a variable
  - substituting the output of a command within a string
- Two ways to perform command substitution
  - placing the command within a set of backquotes
  - placing the command within a set of parenthesis preceded by a `$` sign
- Bash performs the substitution by executing the command and returning the standard output of the command, with any trailing newlines deleted.

Command Substitution

```
$ echo ”The hour is `date +%H`”
The hour is 12
$ set `date`
$ echo $*
Mon Sep 13 12:25:46 AST 2014
$ echo $2 $6
Sep 2014
$ d=$(date)
$ echo $d
Mon Sep 13 12:27:40 AST 2014
$ lines=$(cat file1)
$ echo The time is $(date +%H)
The time is 12
$ machine=$(uname -n)
$ echo $machine
raad2.qatar.tamu.edu
$ pwd
/panfs/vol/f/fachaud74
$ dirname=”$(basename $(pwd))”
$ echo $dirname
fachaud74
```

Shell Scripts

Steps in Creating a Shell Script

- First line should start with **`#!/bin/bash`**
- Comments within the script are preceded by **`#`**

```
#!/bin/bash
# The name of this script is "greetings"

echo "Hello $LOGNAME, it's nice talking to you."
echo "Your present working directory is `pwd`."
echo "You are working on a machine called `uname -n`."
echo "Here is a list of your files."
ls      # list files in the present working directory
echo  "Bye for now $LOGNAME. The time is `date +%T`! "
```

Steps in Running a Shell Script

- Make the script executable with **`chmod u+x`**

```
$ chmod u+x greetings
$ ./greetings
Hello fachaud74, it's nice talking to you.
Your present working directory is /lustre/home/fachaud74.
You are working on a machine called raad2.qatar.tamu.edu.
Here is a list of your files.
abaqus_files       fluent.users  myHomeDir.tar    tmp
fluent-unique.txt  helloDir      README           verifyOLD
fluent-use1.txt    man           script-ex1.bash
Bye for now fachaud74. The time is 15:08:56!
```

جامعة حمد بن خليفة
HAMAD BIN KHALIFA UNIVERSITY
عضو في مؤسسة قطر
Member of Qatar Foundation

شكرًا

Thank you

Questions & discussion?