MAE 5032 High Performance Computing: Methods and Applications

Lecture 2: Unix/Linux – part 1

Ju Liu
Department of Mechanics and Aerospace Engineering
liuj36@sustech.edu.cn

Unix and Linux

UNIX
In early days, every computer has a different operating system.

In 1969, a team of developers in Bell Labs started working on this problem.

They develop a new OS (UNIX):

- Simple and elegant
- Written in C language insteady of assembly code
- Able to be recycled (meaning change only the kernel)

Inside view of the Bell Lab

Unix and Linux

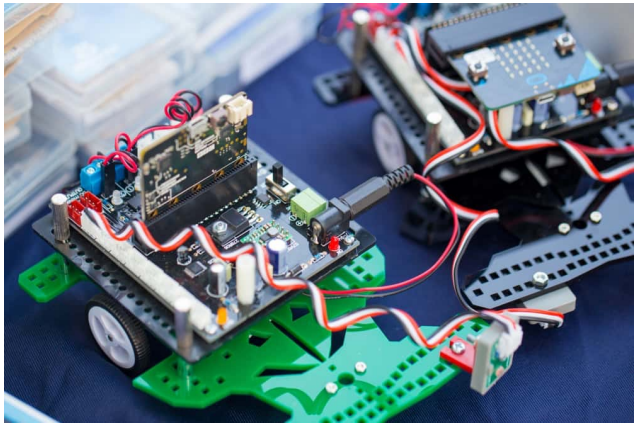
UNIX

Core components:

- **Kernel**: mangages hardware resources
- **Shell**: the interface between the user and the system
- **File system**: employs a tree-like structure, including regular files, directories, and device files
- **Utilities**: tools for text processing, system administration, etc.

Major variants: Linux, Linux, etc.

Application: web servers, data servers, embedded devices like routers
suitable for software development



Unix and Linux

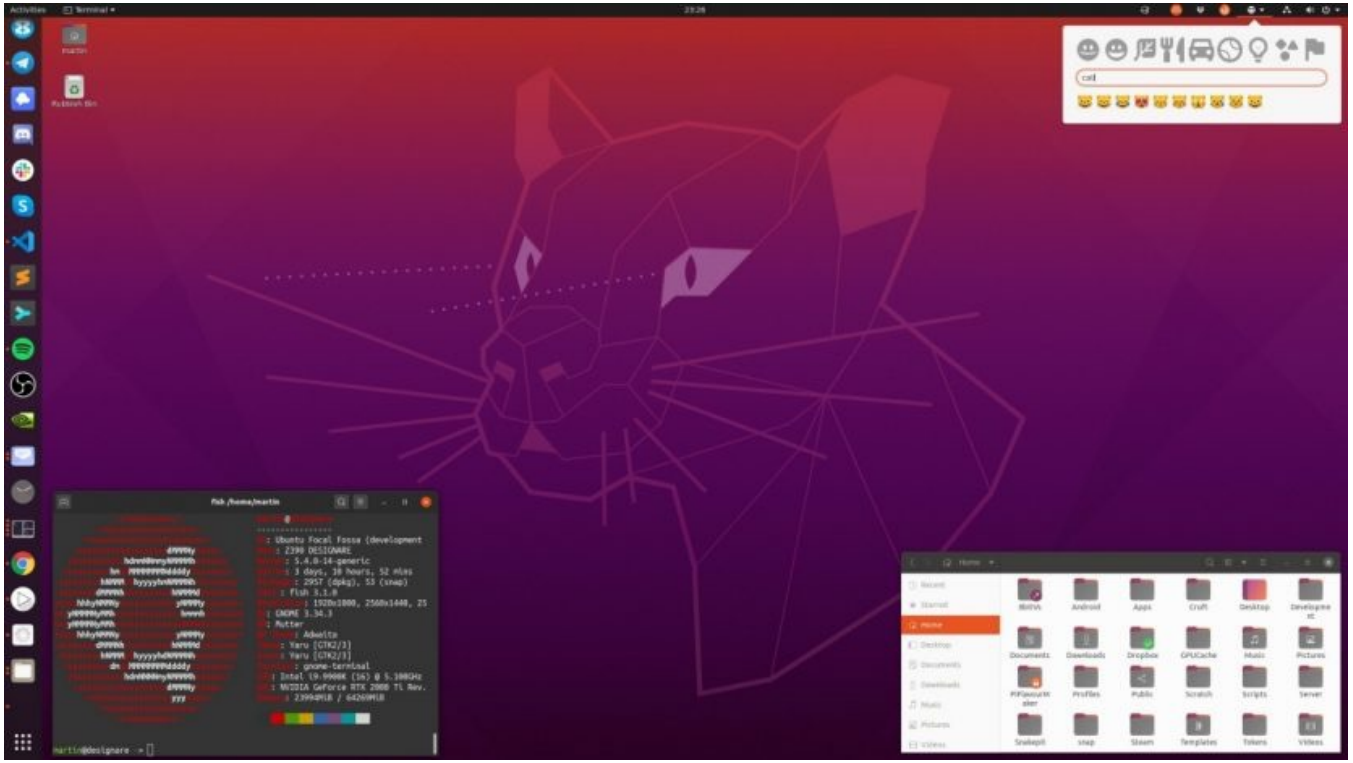
Linus Torvalds, a young man studying computer science at the university of Helsinki, thought it would be a good idea to have some sort of freely available academic version of UNIX, and promptly start to code.

- July 1991 : Linus Torvalds from Helsinki started his hobby: Linux
- Oct. 1991 : version 0.02
- 2002 : version 2.2
- Today : look at www.kernel.org

Linux is an ideal operating system for programmers (everything a good programmer can wish is available : compilers, libraries, …)

Linux for non-experienced users

- Linux has distributions like Windows : RedHat, Scientific Linux, Ubuntu, etc.
- Ubuntu has beautiful graphic user interface.
- Each distribution contains application softwares
- Office : openoffice
- IE : Mozilla/Chrome
- etc.



GNU project

The GNU project was lunched in 1984 with the goal of developing a completely UNIX style operating system which is free.

- GNU General Public Licence (GPL)

A list of GNU software:

- GCC : The GNU C Compiler
- Gnome : The GNU desktop environment
- Emacs : A text editor
- The Gimp : GNU Image Manipulation Program
- In 2012, a fork of Linux kernel became part of GNU project.

GNU sounds like gnu (connochaetes)

Open Source

Open Source means:

User can see binary and source

Users are encouraged to read and understand the source code

Users can debug and recompile source

Closed Source means:

Uers are only given binaries

Uers cannot read the source

Uers cannot recompile the code

Linux

Advantages:

- Linux is free and more secure than Windows
- Open sourced
- Necessary for HPC
- Rare for a Linux system to slow down or crashes
- Easy to debug codes

Disadvantages:

- Not very user friendly and learning curve can be sharp for beginners
- Is an open-sourced product trustworthy?

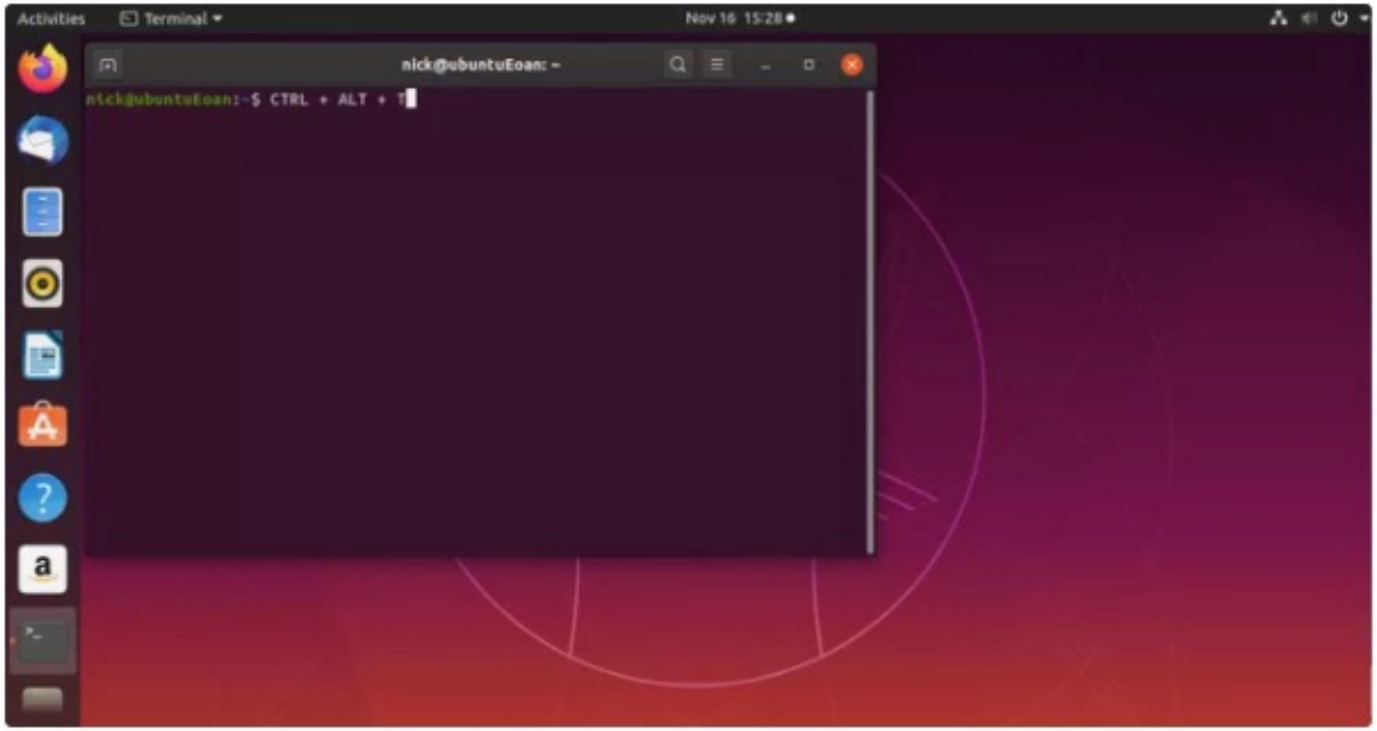
Outline

1. Basic Commands
2. File attributes and permissions
3. Regular expressions
4. Interacting with the shell
5. Unix pipes
6. Job control
7. UNIX Environment variables
8. Text Editors
9. Shell scripting

Open a terminal

01 Open a Linux Terminal Using Ctrl+Alt+T

of 05



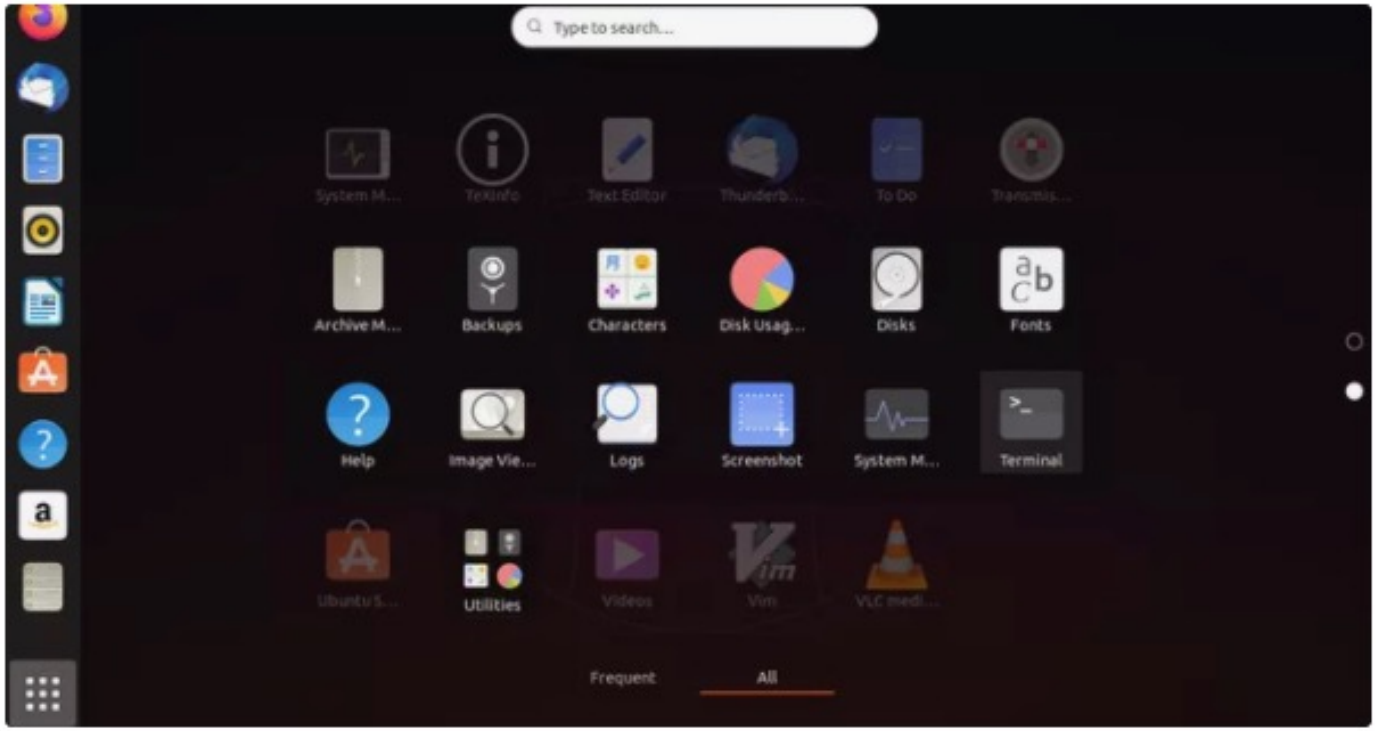
Open a terminal

02 of 05 Search Using the Ubuntu Dash

Open a terminal

03 Navigate the GNOME App Launcher

of 05



Basic Unix Commands

Unix Command	what it does
`ls`	list directory contents
`cd`	change directory
`df`	display free disk space
`cat`	concatenate and print files
`more (less)`	a filter for paging through text one screenful at a time.
`head`	display first lines of a file
`pwd`	return working directory name
`cp`	copy files
`awk`	a pattern-directed scanning and processing language

Basic Unix Commands

Unix Command	Function
`rm`	remove directory entries
`chmod`	change file modes or Access Control Lists
`tail`	display the last part of a file
`touch`	change file access and modification times
`mkdir`	make directories
`rmdir`	remove directories
`find`	walk a file hierarchy
`grep`	print lines matching a pattern
`chown/chgrp`	change file owner and group

`ls`: list directory contents

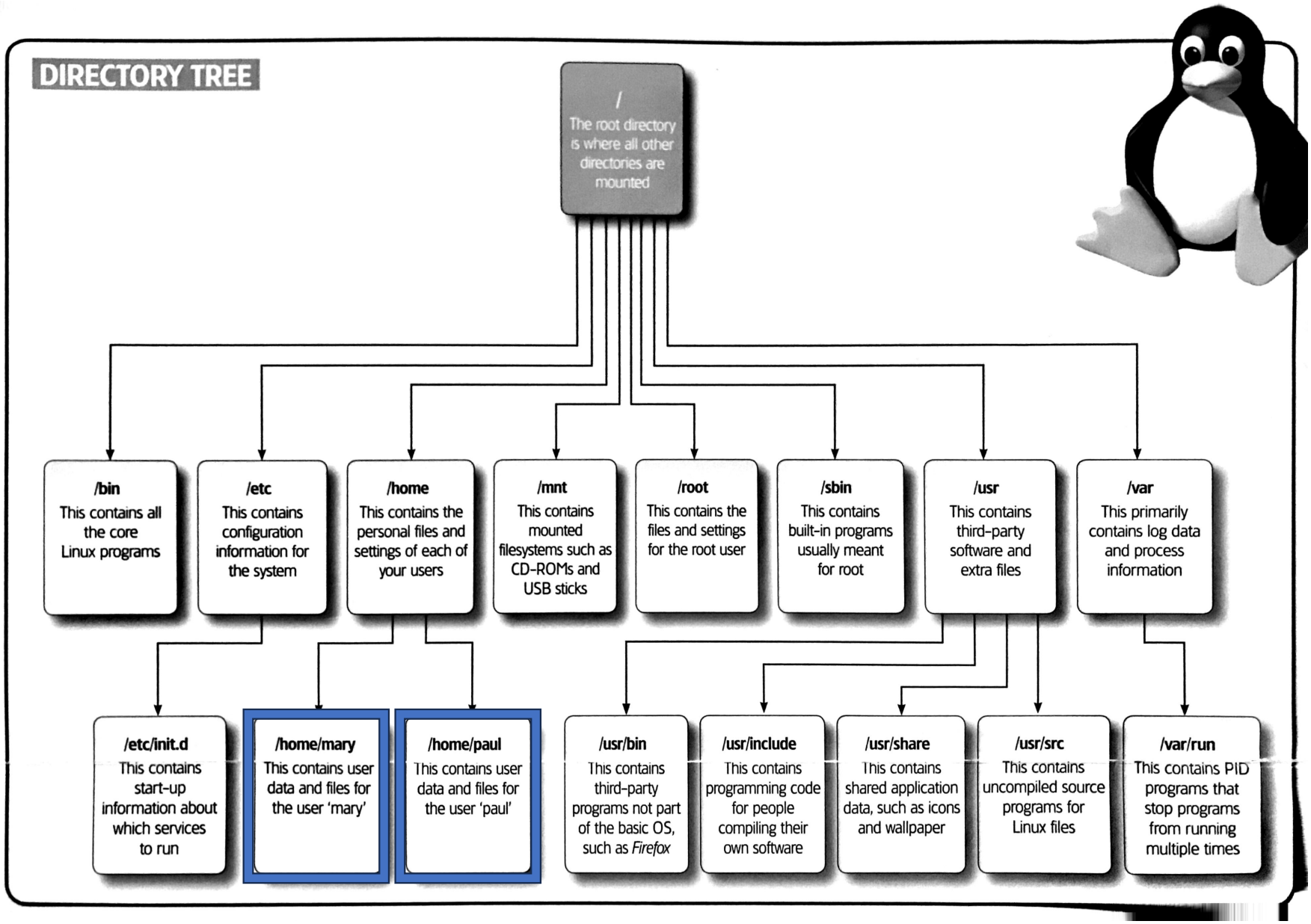
```
ls [-options] [names]
```

- The brackets around **`options`** and **`names`** in the general form of the **`ls`** command means that something is optional
- This type of description is common in the documentation for Unix commands
- To use command line options precede the option letter(s) with a hyphen
- The **`ls`** command supports many options

`ls`	list files in current directory
`ls /`	list files in the root directory
`ls .`	list files in the current directory
`ls ..`	list files in the parent directory
`ls /usr`	list files in the directory /usr
`ls -l .`	long format (include file times, owner and permissions)
`ls -a .`	all (shows hidden files as well as regular files)
`ls -F .`	include special char to indicate file types

In Unix, hidden files have names that start with "`.`"

- Absolute Path : path that starts from the root directory (/)
- Relative Path : path that does NOT start from the root directory


DIRECTORY TREE
/
The root directory is where all other directories are mounted
/bin
This contains all the core Linux programs
/etc
This contains configuration information for the system
/home
This contains the personal files and settings of each of your users
/mnt
This contains mounted filesystems such as CD-ROMs and USB sticks
/root
This contains the files and settings for the root user
/sbin
This contains built-in programs usually meant for root
/usr
This contains third-party software and extra files
/var
This primarily contains log data and process information
/etc/init.d
This contains start-up information about which services to run
/home/mary
This contains user data and files for the user 'mary'
/home/paul
This contains user data and files for the user 'paul'
/usr/bin
This contains third-party programs not part of the basic OS, such as Firefox
/usr/include
This contains programming code for people compiling their own software
/usr/share
This contains shared application data, such as icons and wallpaper
/usr/src
This contains uncompiled source programs for Linux files
/var/run
This contains PID programs that stop programs from running multiple times

cd: change directories

```
lslogin$ ls
foo.c       mydata      typescript
lslogin$ cd mydata
lslogin$ pwd
/Users/sunit/workdir/mydata
lslogin$ ls
backup.fasta data.fasta    moredata
lslogin$ cd ..
lslogin$ pwd
/Users/sunit/workdir
```

pwd: print working directory name

"**cd**" with no arguments takes you to your HOME directory.

`cat`: concatenate and print files

```
lslogin$ cat data.fasta
>c01_009 499 amino acids  MW=55632 D  pI=5.38 numambig=0
MPGGFILAIDEGTTSARAIIYNQDLEVLGIGQYDFPQHYPSPGYVEHNPDEIWNAQMLAI
KEAMKKAKIESRQVAGIGVTNQRETTILWDAISGKPIYNAIVWQDRRTSNITDWLKENYF
GMIKDKTGLIPDPYFSGSKIKWILDNLPNVRSKAEKGEIKFGTIDTYLIWKLTNGKIHVT
IGITRGTTKAHIARAILESIAYQNRDVIEIMEKESGTKINILKVDGGGAKDNLLMQFQAD
ILGIRVVRPKVMETASMGVAMLAGLAINYWNSLNELKQKWTVDKEFIPSINKEERERRYN
AWKEAVKRSLGWEKSLGSK*
```

more: display a file's contents one screen at a time

```
lslogin$ more data.fasta
>c01_009 499 amino acids  MW=55632 D  pI=5.38 numambig=0
MPGGFILAIDEGTTSARAIIYNQDLEVLGIGQYDFPQHYPSPGYVEHNPDEIWNAQMLAI
KEAMKKAKIESRQVAGIGVTNQRETTILWDAISGKPIYNAIVWQDRRTSNITDWLKENYF
GMIKDKTGLIPDPYFSGSKIKWILDNLPNVRSKAEKGEIKFGTIDTYLIWKLTNGKIHVT
IGITRGTTKAHIARAILESIAYQNRDVIEIMEKESGTKINILKVDGGGAKDNLLMQFQAD
ILGIRVVRPKVMETASMGVAMLAGLAINYWNSLNELKQKWTVDKEFIPSINKEERERRYN
AWKEAVKRSLGWEKSLGSKLPNVRSKAEKGEIKFGTIDTYLIWKLTNGRDVIEIMEKESG
TKINILKVDGGGAKDNLLMQFQILDNLPNVRSKAEKGEIKFGTIDTYLIWKLTNGTSAIG
AHIARAILESIAYQNRDVIEIMEKESGTKINILKVDGGGAKDNLLMQFQADTDWLKENYF
LGWEKSLGSKLPNVRSKAEKGEIKFGTIDTYLIWKLTNGRDVIEIMIKFGTIDTYLIWTG
GMIKDKTGLIPDPYFSGSKIKWILDNLPNVRSKAEKGEIKFGTIDTYLIWKLTNGKIHVT
IGITRGTTKAHIARAILESIAYQNRDVIEIMEKESGTKINILKVDGGGAKDNLLMQFQAD
DKTGLIPDPYFSGSKIKWILDNLPNVRSKAEKGEIGVAMLAGLAINYWNSLNELKFKIHE
NILKVDGGGAKDNLLMQFQILDNLPNVRSKAEKGEIKFGTIDTYLFQFQILDNLPNVRSK
AEKGEIKFGTIDTYLIWILDVIEIMEKESGTKINILKVDGGGKVDGGIAYQNRDVIEIME
LWDAISGKPIYNDNLPNVRSKAEKGEIGVAMLAGLAINNLPNVRSKAEKGEIGKGEIGVA
--More--(75%)
```

Notes: hit <space> to see the next page
hit "q" to quit, "/" to search, read the man page.
"less" is an enhanced version of "more" on Linux

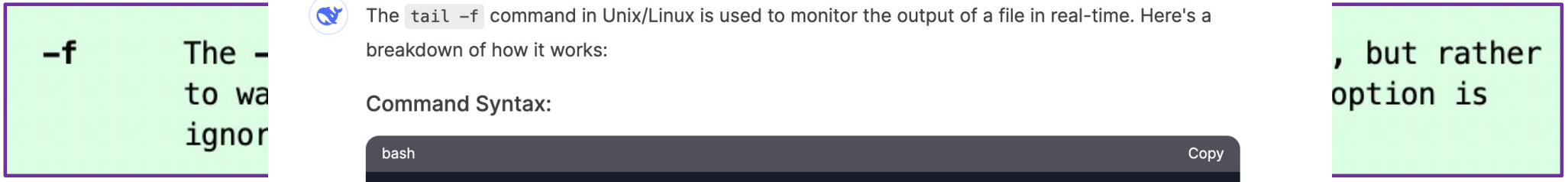
`head` & `tail`: display first or last lines of a file

- head displays the top of a file
  - head -n displays the top n lines
  - default is 10
- tail displays the bottom of a file
  - tail -n displays the bottom n lines
  - default is 10
  - tail +n displays the file starting at line n

```
tail –f output_file
```

`head` & `tail`: display first or last lines of a file

explain tail -f outputfile to me



The `tail -f` command in Unix/Linux is used to monitor the output of a file in real-time. Here's a breakdown of how it works:

Command Syntax:

bash Copy

```
tail -f outputfile
```

Explanation:

1. `tail`:
   - The `tail` command is used to display the last part of a file (by default, the last 10 lines).
2. `-f` **(follow)**:
   - The `-f` option tells `tail` to "follow" the file. Instead of just displaying the last few lines and exiting, it keeps the file open and continuously prints new lines as they are appended to the file.
3. `outputfile`:
   - This is the name of the file you want to monitor. It could be a log file, a data file, or any other text file that is being updated in real-time.

`mkdir`: make directories

```
lslogin$ ls
my_data      data.fasta      backup.fasta      prog
lslogin$ ls -F
my_data/      data.fasta      backup.fasta      prog*
lslogin$ alias ls 'ls -F'
lslogin$ ls
my_data/      data.fasta      backup.fasta      prog*
lslogin$ mkdir new_data
lslogin$ ls
my_data/      new_data/      data.fasta      backup.fasta    prog*
```

cp: copy files

```
lslogin$ ls
bin/            my_data/          data.fasta
lslogin$ cp data.fasta backup.fasta
lslogin$ ls
bin/            my_data/          data.fasta      backup.fasta
lslogin$
```

copy one or multiple files to a directory : cp [options] source1 source2 source3 … directory

copy a file : cp source destination

cp: copy files

```
lslogin$ ls
bin/            my_data/          data.fasta
lslogin$ cp data.fasta backup.fasta
lslogin$ ls
bin/            my_data/          data.fasta      backup.fasta
lslogin$
```

use the –r option to copy recursively (for folders)

mv: move or rename files

```
lslogin$ ls
bin/            my_data/          data.fasta      backup.fasta
lslogin$ mv backup.fasta my_data
lslogin$ ls
bin/            my_data/          data.fasta
lslogin$ ls my_data
backup.fasta
lslogin$ mv data.fasta Dec15.fasta
lslogin$ ls
bin/            my_data/          Dec15.fast
```

mv [options] source destination

if source and destination are in the same folder, mv changes the name

`rm`: deletes files - *permanently*

```
lslogin$ ls
bin/            my_data/          data.fasta      backup.fasta
lslogin$ rm backup.fasta
lslogin$ ls
bin/            my_data/          data.fasta
```

`rm -i` : interactive, ask for confirmation
`rm -r` : recursively remove directories
`rm -f` : override lack of write permission

Note: There is no Recycle bin or Undelete Key!
Thou shalt know what thou are doing...

UNIX Commands: groups

```
lslogin$ id
uid=1003(rob) gid=10(staff)

lslogin$ groups
staff sysadmin wwwdevel
```

Use **groups** to see all the group names to which you belong.

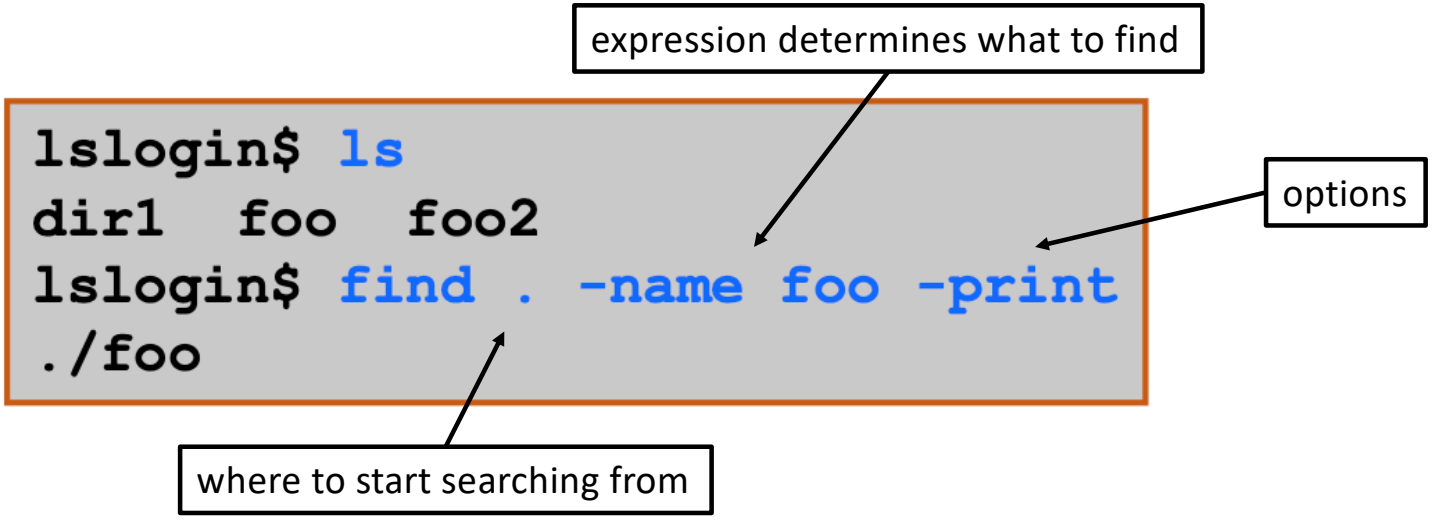
grep: extracts lines from a file that match a given string or pattern

```
lslogin$ cat sequence.fas
>c01_009 499 amino acids  MW=55632 D  pI=5.38 numambig=0
MPGGFILAIDEGTTSARAIIYNQDLEVLGIGQYDFPQHYPSPGYVEHNPDEIWNAQMLAI
KEAMKKAKIESRQVAGIGVTNQRETTILWDAISGKPIYNAIVWQDRRTSNITDWLKENYF
DYSNASRTMLFNINKLEWDREILELLKIPESILPEVRPSSDIYGYTEVLGSSIPISGDAG
DQQAALFGQVAYDMGEVKSTYGTGSFILMNIGSNPIFSENLLTTIAWGLESKRVTYALEG
SIFITGAAVQWFRDGLGREPKICKSBUTTASVPDTGGVYFVPAFVGLGAPYWDPYARGLI
IGITRGTTKAHIARAILESIAYQNRDVIEIMEKESGTKINILKVDGGGAKDNLLMQFQAD
ILGIRVVRPKVMETASMGVAMLAGLAINYWNSLNELKQKWTVDKEFIPSINKEERERRYN
AWKEAVKRSLGWEKSLGSK*

lslogin$ grep AA sequence.fas  Search in sequence.fas file for AA
DQQAALFGQVAYDMGEVKSTYGTGSFILMNIGSNPIFSENLLTTIAWGLESKRVTYALEG
SIFITGAAVQWFRDGLGREPKICKSBUTTASVPDTGGVYFVPAFVGLGAPYWDPYARGLI
```

`find`: walk a file hierarchy

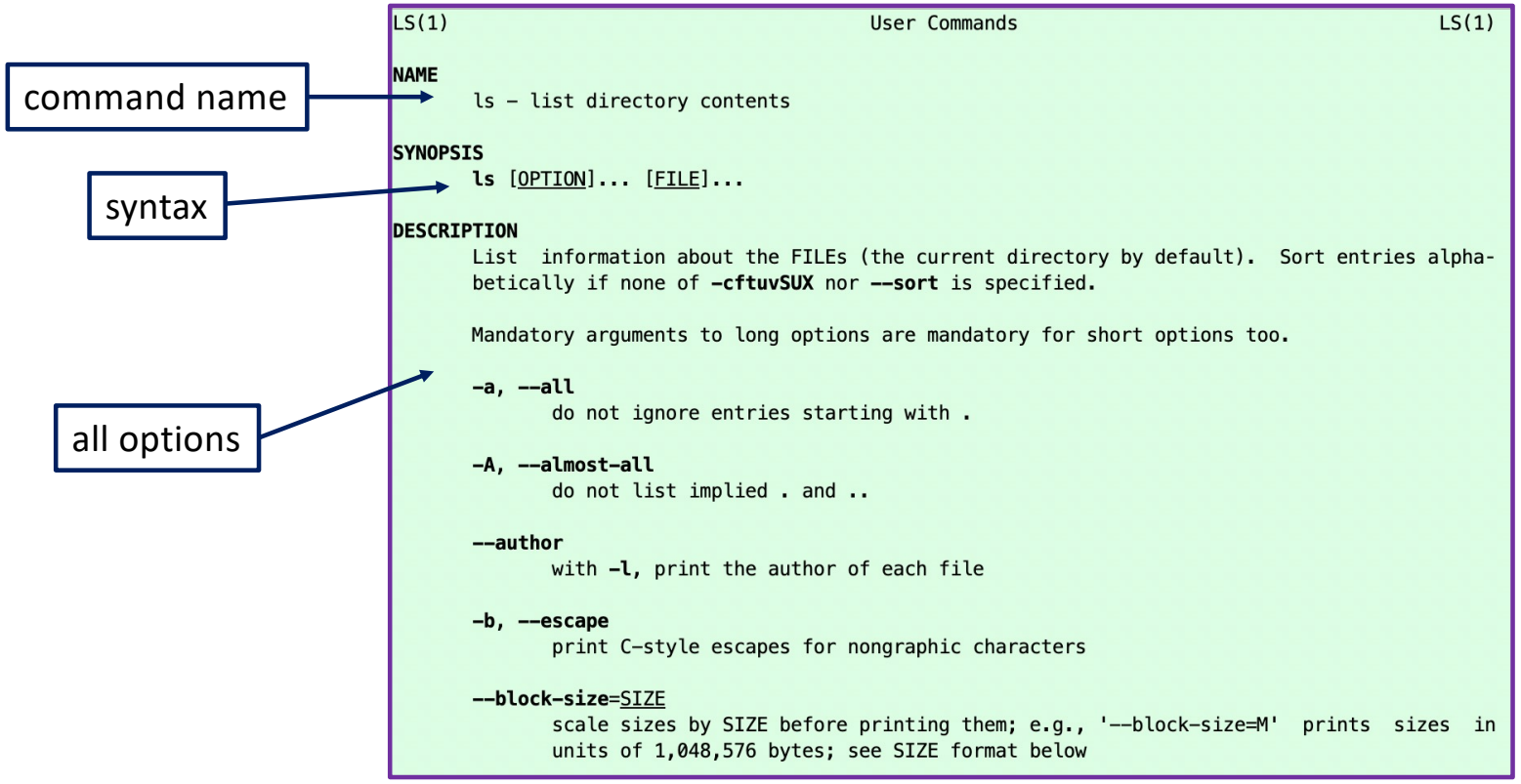
- At its simplest, find searches the filesystem for files whose name matches a specific pattern
- However, it can do a lot more and is one of the most useful commands in Unix (as it can find specific files and then perform operations on them)



man: manual of Linux

man [command]

display the whole manual page of a particular command



Hit enter to page down
Hit q to exit

Outline

1. Basic Commands
2. File attributes and permissions
3. Regular expressions
4. Interacting with the shell
5. Unix pipes
6. Job control
7. UNIX Environment variables
8. Text Editors
9. Shell scripting

Unix/Linux accounts

- To access a Unix/Linux system, you need to have an account
- The account includes:
  - Username and password
  - Userid (UID) and groupid (GID)
    - An integer number
  - Home directory (e.g. /home/username)
    - a place to keep your files
    - May have a quota (the system impose a limit on how much data you can have)
  - a default shell preference

File contents

- A file is a basic unit of storage
- Unix/Linux file names can contain any characters
- Each file can hold some raw data
  - Unix/Linux does not impose any structure on files
  - Text file, sequence of integers, database records, etc.
  - In scientific computing, we often use binary files for efficiency in storage and data access
  - Portability is an issue (little endian vs. big endian)
  - We use formats like HDF to handle the data

Directories

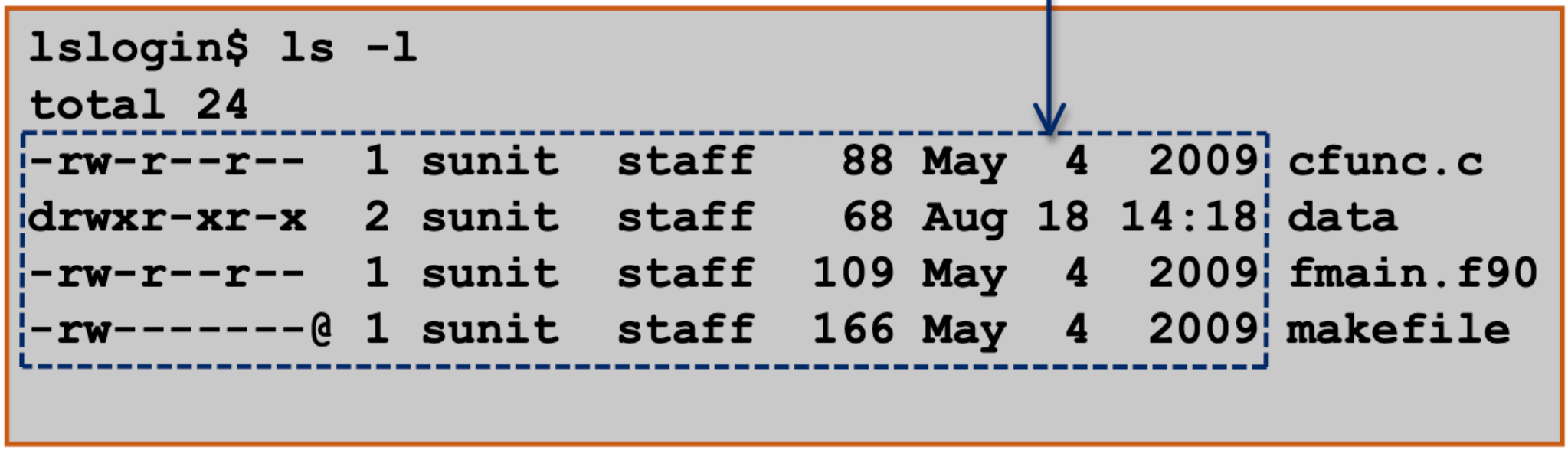
- A directory is a special kind of file
- Unix/Linux uses a directory to hold information about other files
- We often think of a directory as a container that holds other files (or directories).
- Windows users can relate a directory to the same idea as a folder.

Pathnames

- Each file has a pathname (we talked about this earlier)
- The full pathname of a file includes the file name and the name of the directory that holds it, and the name of the directory that holds the directory, ……
- You may start the root with “/” and follow the path down the hierarchy, or you may start from your current working directory
- Use “/” between every directory
- Special directory names: . .. ~

File Attributes

The **`ls -l`** command (long listing) displays specific attributes about each file or directory.



```
lslogin$ ls -l
total 24
-rw-r--r--   1 sunit   staff    88 May   4   2009 cfunc.c
drwxr-xr-x   2 sunit   staff    68 Aug 18 14:18 data
-rw-r--r--   1 sunit   staff   109 May   4   2009 fmain.f90
-rw-------@ 1 sunit   staff   166 May   4   2009 makefile
```

File Attributes cont.

Every file has a specific list of attributes:

- Access Times
  - when the file was created
  - when the file was last changed
  - when the file was last read
- Size
- Owners
  - user (UID)
  - group (GID)
- Permissions

File Time Attributes

- Time Attributes

ls -l <filename>	when the file was last changed
ls -lc <filename>	when the file was created

```
-> stat /
  File: '/'
  Size: 4096            Blocks: 8          IO Block: 4096   directory
Device: fd00h/64768d    Inode: 128         Links: 24
Access: (0555/dr-xr-xr-x)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2022-02-15 15:11:56.462507715 +0800
Modify: 2022-01-28 01:23:22.509320161 +0800
Change: 2022-01-28 01:23:22.509320161 +0800
 Birth: -
```

- Display date-related attributes with the **stat** command

File Permissions

- Each file has a set of permissions that control who can access the file
- There are three different types of permissions:
  - read abbreviated r
  - write abbreviated w
  - execute abbreviated x
- In Unix, there are permission levels associated with three types of people that might access a file:
  - owner (you)
  - group (a group of other users that you set up)
  - world (anyone else browsing around on the file system)

File Permissions Display Format

The first column specifies the type of file:

Symbol	File type	Example
-	plain file	
d	directory	
c	character device	
b	block device	
l	symbolic link	

File Permissions Display Format cont.

- Meaning for Files:
  - r - allowed to read
  - w - allowed to write
  - x - allowed to execute
- Meaning for Directories:
  - r - allowed to see the names of the files
  - w - allowed to add and remove files
  - x - allowed to enter the directory

Changing File Permissions

- The **chmod** command changes the permissions associated with a file or directory
- Basic syntax is:

```
chmod mode file
```

- The **mode** can be specified in two ways:
  - symbolic representation
  - octal number
- Both methods achieve the same result (user's choice)
- Multiple symbolic operations can be given, separated by commas

`chmod`: Symbolic Representation

- *Symbolic* Mode representation has the following form:

`[ugoa] [+-=] [rwxX…]`

`u`=user	**`+`** add permission	**`r`**=read
`g`=group	**`-`** remove permission	**`w`**=write
`o`=other	**`=`** set permission	**`x`**=execute
`a`= all		**`X`**= see below

- The **`X`** permission option is very handy - it sets to execute only if the file is a directory or already has execute permission

chmod: Symbolic Mode Examples

```
lslogin$ ls -al foo
-rw-------  1 karl support ...
```

```
lslogin$ chmod g=rw foo
lslogin$ ls -al foo
-rw-rw----  1 karl support ...
```

```
lslogin$ chmod u-w,g+x,o=x foo
lslogin$ ls -al foo
-r--rwx--x  1 karl support ...
```

chmod: Octal Representation

- Octal Mode uses a single argument string which describes the permissions for a file (3 digits)
- Each digit of this number is a code for each of the three permission levels (user, group, world)
- Permissions are set according to the following numbers:
  - Read = 4
  - Write = 2
  - Execute = 1

	Permission Level
0	no permissions
1	execute only
2	write only
3	write and execute (1+2)
4	read only
5	read and execute (4+1)
6	read and write (4+2)
7	read, write and execute (4+2+1)

- Sum the individual permissions to get the desired combination

umask: set default permission

- umask is used to set default permissions for files and directories

```
$ umask          # display current value (as octal)
0022
$ umask -S       # display current value symbolically
u=rwx,g=rx,o=rx
```

umask: set default permission

- umask is used to set default permissions for files and directories

```
$ umask 007     # set the mask to 007
$ umask         # display the mask (in octal)
0007            #   0 - special permissions (setuid | setgid | sticky )
                #   0 - (u)ser/owner part of mask
                #   0 - (g)roup part of mask
                #   7 - (o)thers/not-in-group part of mask
$ umask -S      # display the mask symbolically
u=rwx,g=rwx,o=
```

Outline

1. File attributes and permissions
2. Basic Commands
3. Regular expressions
4. Interacting with the shell
5. Unix pipes
6. Job control
7. UNIX Environment variables
8. Text Editors
9. Shell scripting

grep: extracts lines from a file that match a given string or pattern

Options	Description
`-i`	Ignore case distinctions on Linux and Unix
`-w`	Force PATTERN to match only whole words
`-v`	Select non-matching lines
`-n`	Print line number with output lines
`-h`	Suppress the Unix file name prefix on output
`-r`	Search directories recursivly on Linux
`-R`	Just like -r but follow all symlinks
`-l`	Print only names of FILEs with selected lines
`-c`	Print only a count of selected lines per FILE
`--color`	Display matched pattern in colors

Examples:

grep –r dander /home/dan

grep –n danger example.txt

grep –i DanGer example.txt

grep: extracts lines from a file that match a given string or pattern

Usage of quotes

1. grep hello world file1 will look for word “hello” in files named world and file1.
2. grep “hello world” file1 will look for “hello world” in file1.

The choice between single and double quotes is only important if the search string contains variables to be evaluated.

```
VAR="serverfault"
grep '$VAR' file1
grep "$VAR" file1
```

look for $VAR

look for serverfault

Regular Expressions

grep : get regular expression.

grep understands basic regular expressions.

egrep understands extended regular expressions.

```
egrep –ni –color “some string pattern” file-to-search
```

Regular Expressions

- In addition to grep, a number of Unix commands support the use of regular expressions to describe patterns:
  - sed
  - awk
  - perl

  Examples: runoo+b matches runoob, runooob, runooooooob, etc.
  runoo*b matches runob, runoob, runooooob, etc.
  colou?r matches color or colour.
- General search pattern characters:
  - Any character (except a metacharacter) matches itself
  - "." matches any character except a newline
  - "*" matches zero or more occurrences of the single preceding character
  - "+" matches one or more of the proceeding character
  - "?" matches zero or one of the proceeding character
- Additional special characters:
  - "()" parentheses are used to quantify a sequence of characters
  - "|" works as an OR operator
  - "{}" braces are used to indicate ranges in the number of occurrences

Regular Expressions

- In addition to grep, a number of Unix commands support the use of regular expressions to describe patterns:
  - sed
  - awk
  - perl

  Examples: “g(la|oo)d” will match glad or good
  “go{1,2}d” will match god or good
- General search pattern characters:
  - Any character (except a metacharacter) matches itself
  - “.” matches any character except a newline
  - “*” matches zero or more occurrences of the single preceding character
  - “+” matches one or more of the proceeding character
  - “?” matches zero or one of the proceeding character
- Additional special characters:
  - “()” parentheses are used to quantify a sequence of characters
  - “|” works as an OR operator
  - “{}” braces are used to indicate ranges in the number of occurrences

Regular Expressions

```
-> cat regular_express.txt
"Open Source" is a good mechanism to develop programs
apple is my favorite food
Football game does not use feet only
Oh! The soup taste good ^M
The symbol '*' is represented as start
<Happy>
Goooogle is good!
Go Go Go
However, this dress is about $ 333 dollars ^L.
\\ This is commen.t
# End of file
```

Regular Expressions

```
-> cat regular_express.txt
"Open Source" is a good mechanism to develop programs
apple is my favorite food
Football game does not use feet only
Oh! The soup taste good ^M
The symbol '*' is represented as start
<Happy>
Goooogle is good!
Go Go Go
However, this dress is about $ 333 dollars ^L.
\\ This is commen.t
# End of file
```

```
-> grep -n 'the' regular_express.txt
```

Regular Expressions

```
-> cat regular_express.txt
"Open Source" is a good mechanism to develop programs
apple is my favorite food
Football game does not use feet only
Oh! The soup taste good ^M
The symbol '*' is represented as start
<Happy>
Goooogle is good!
Go Go Go
However, this dress is about $ 333 dollars ^L.
\\ This is commen.t
# End of file
```

```
-> grep -ni 'the' regular_express.txt
4:Oh! The soup taste good ^M
5:The symbol '*' is represented as start
```

Regular Expressions

```
-> cat regular_express.txt
"Open Source" is a good mechanism to develop programs
apple is my favorite food
Football game does not use feet only
Oh! The soup taste good ^M
The symbol '*' is represented as start
<Happy>
Goooogle is good!
Go Go Go
However, this dress is about $ 333 dollars ^L.
\\ This is commen.t
# End of file
```

```
-> grep -n --color 'g.o' regular_express.txt
1:"Open Source" is a good mechanism to develop programs
4:Oh! The soup taste good ^M
7:Goooogle is good!
```

Regular Expressions

```
-> cat regular_express.txt
"Open Source" is a good mechanism to develop programs
apple is my favorite food
Football game does not use feet only
Oh! The soup taste good ^M
The symbol '*' is represented as start
<Happy>
Goooogle is good!
Go Go Go
However, this dress is about $ 333 dollars ^L.
\\ This is commen.t
# End of file
```

```
-> grep -n --color 'go*' regular_express.txt
1:"Open Source" is a good mechanism to develop programs
3:Football game does not use feet only
4:Oh! The soup taste good ^M
7:Goooogle is good!
```

Regular Expressions

Regular Expressions

- If you really want to match a period '.', you need to escape it with a backslash "\."

Regexp	Matches	Does not match
a.b	axb	abc
a\.b	a.b	axb

Regular Expressions

```
-> cat regular_express.txt
"Open Source" is a good mechanism to develop programs
apple is my favorite food
Football game does not use feet only
Oh! The soup taste good ^M
The symbol '*' is represented as start
<Happy>
Goooogle is good!
Go Go Go
However, this dress is about $ 333 dollars ^L.
\\ This is commen.t
# End of file
```

```
-> grep -n --color 'n\.t' regular_express.txt
10:\\ This is commen.t
```

Regular Expressions

- A character class, also called a character set can be used to match only one out of several characters
- To use, simply place the characters you want to match between square brackets []
- You can use a hyphen inside a character class to specify a range of characters
- Placing a caret (^) after the opening square bracket will negate the character class. The result is that the character class will match any character that is not in the character class
- Examples:
  - [abc] matches a single a b or c
  - [0-9] matches a single digit between 0 and 9
  - [^A-Za-z] matches a single character as long as it is not a letter

Regular Expressions

```
-> cat regular_express.txt
"Open Source" is a good mechanism to develop programs
apple is my favorite food
Football game does not use feet only
Oh! The soup taste good ^M
The symbol '*' is represented as start
<Happy>
Goooogle is good!
Go Go Go
However, this dress is about $ 333 dollars ^L.
\\ This is commen.t
# End of file
```

```
-> grep -n --color '[a-z]es' regular_express.txt
3:Football game does not use feet only
5:The symbol '*' is represented as start
9:However, this dress is about $ 333 dollars ^L.
```

Regular Expressions

```
-> cat regular_express.txt
"Open Source" is a good mechanism to develop programs
apple is my favorite food
Football game does not use feet only
Oh! The soup taste good ^M
The symbol '*' is represented as start
<Happy>
Goooogle is good!
Go Go Go
However, this dress is about $ 333 dollars ^L.
\\ This is commen.t
# End of file
```

```
-> grep -n --color '[^l-q]es' regular_express.txt
5:The symbol '*' is represented as start
9:However, this dress is about $ 333 dollars ^L.
```

grep: cont.

```
lslogin$ cat sequence.fas
>c01_009 499 amino acids  MW=55632 D  pI=5.38 numambig=0
MPGGFILAIDEGTTSARAIIYNQDLEVLGIGQYDFPQHYPSPGYVEHNPDEIWNAQMLAI
KEAMKKAKIESRQVAGIGVTNQRETTILWDAISGKPIYNAIVWQDRRTSNITDWLKENYF
DYSNASRTMLFNINKLEWDREILELLKIPESILPEVRPSSDIYGYTEVLGSSIPISGDAG
DQQAALFGQVAYDMGEVKSTYGTGSFILMNIGSNPIFSENLLTTIAWGLESKRVTYALEG
SIFITGAAVQWFRDGLGREPKICKSBUTTASVPDTGGVYFVPAFVGLGAPYWDPYARGLI
IGITRGTTKAHIARAILESIAYQNRDVIEIMEKESGTKINILKVDGGGAKDNLLMQFQAD
ILGIRVVRPKVMETASMGVAMLAGLAINYWNSLNELKQKWTVDKEFIPSINKEERERRYN
AWKEAVKRSLGWEKSLGSK*

lslogin$ grep '[ST].[RK]' sequence.fas
MPGGFILAIDEGTTSARAIIYNQDLEVLGIGQYDFPQHYPSPGYVEHNPDEIWNAQMLAI
KEAMKKAKIESRQVAGIGVTNQRETTILWDAISGKPIYNAIVWQDRRTSNITDWLKENYF
DQQAALFGQVAYDMGEVKSTYGTGSFILMNIGSNPIFSENLLTTIAWGLESKRVTYALEG
IGITRGTTKAHIARAILESIAYQNRDVIEIMEKESGTKINILKVDGGGAKDNLLMQFQAD
```

a regular expression search

Regular Expressions

- Since certain character classes are used often, a series of shorthand character classes are available for convenience:

  \d a digit. eg [0-9]
  \D a non-digit, e.g.. [^0-9]
  \w a word character (matches letters and digits)
  \W a non-word character
  \s a whitespace character
  \S a non-whitespace character

```
-> grep -n --color '\d' regular_express.txt
9:However, this dress is about $ 333 dollars ^L.
```

Regular Expressions

- Since certain character classes are used often, a series of shorthand character classes are

```
-> grep -n --color '\W' regular_express.txt
1:"Open Source" is a good mechanism to develop programs
2:apple is my favorite food
3:Football game does not use feet only
4:Oh! The soup taste good ^M
5:The symbol '*' is represented as start
6:<Happy>
7:Goooogle is good!
8:Go Go Go
9:However, this dress is about $ 333 dollars ^L.
10:\\ This is commen.t
11:# End of file
```

Regular Expressions

Regular Expressions

- More shorthand classes are available for matching boundaries:

```
-> grep -n --color "^H" regular_express.txt
9:However, this dress is about $ 333 dollars ^L.
```

^ matches the beginning of a string or line

$ matches the end of a string or line

\b word boundary

Regular Expressions

Regular Expressions

- More shorthand classes are available for matching boundaries:

```
-> grep -n --color "y$" regular_express.txt
3:Football game does not use feet only
```

^ matches the beginning of a string or line

$ matches the end of a string or line

\b word boundary

Regular Expressions

Regular Expressions Examples

- “notice” a string that has the text "notice" in it
- “F.” matches an “F” followed by any character
- “a.b” matches “a” followed by any 1 char followed by “b”
- “^The” matches any string that starts with "The"
- “oh boy$” matches a string that ends in the substring "oh boy";
- “^abc$” matches a string that starts and ends with "abc" -- that could only be "abc" itself!
- “ab*” matches an “a” followed by zero or more “b”'s ("a", "ab", "abbb", etc.)
- “ab+” similar to previous, but there's at least one “b” ("ab", "abbb", etc.)
- “(b|cd)ef” matches a string that has either "bef" or "cdef"
- “a(bc)*” matches an *“a”* followed by zero or more copies of the sequence "bc"
- “ab{3,5}” matches an *“a”* followed by three to five *“b”*'s ("abbb", "abbbb", or "abbbbb")
- “[Dd][Aa][Vv][Ee]” matches "Dave" or "dave" or "dAVE“, does not match "ave" or "da"

Regular Expressions

Search for a cell phone number:

```
-> egrep -n --color "\b1[3|4|5|7|8][0-9]{9}\b"
```

Search for an email address:

```
-> egrep -n --color "\b\w*@\w*" demo
```

Search on the stack overflow website for more sophisticated ways.

Could be a job interview question for programming jobs.

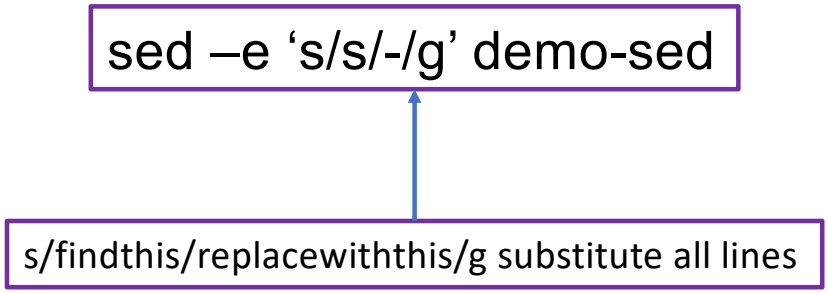
Regular Expressions

sed : stream editor

Modify a stream of data in some fashion

The following command will substitute all “s” in the demo-sed file by “-”

Read Chapter 10 of the book “Unix in a nutshell” for more details.



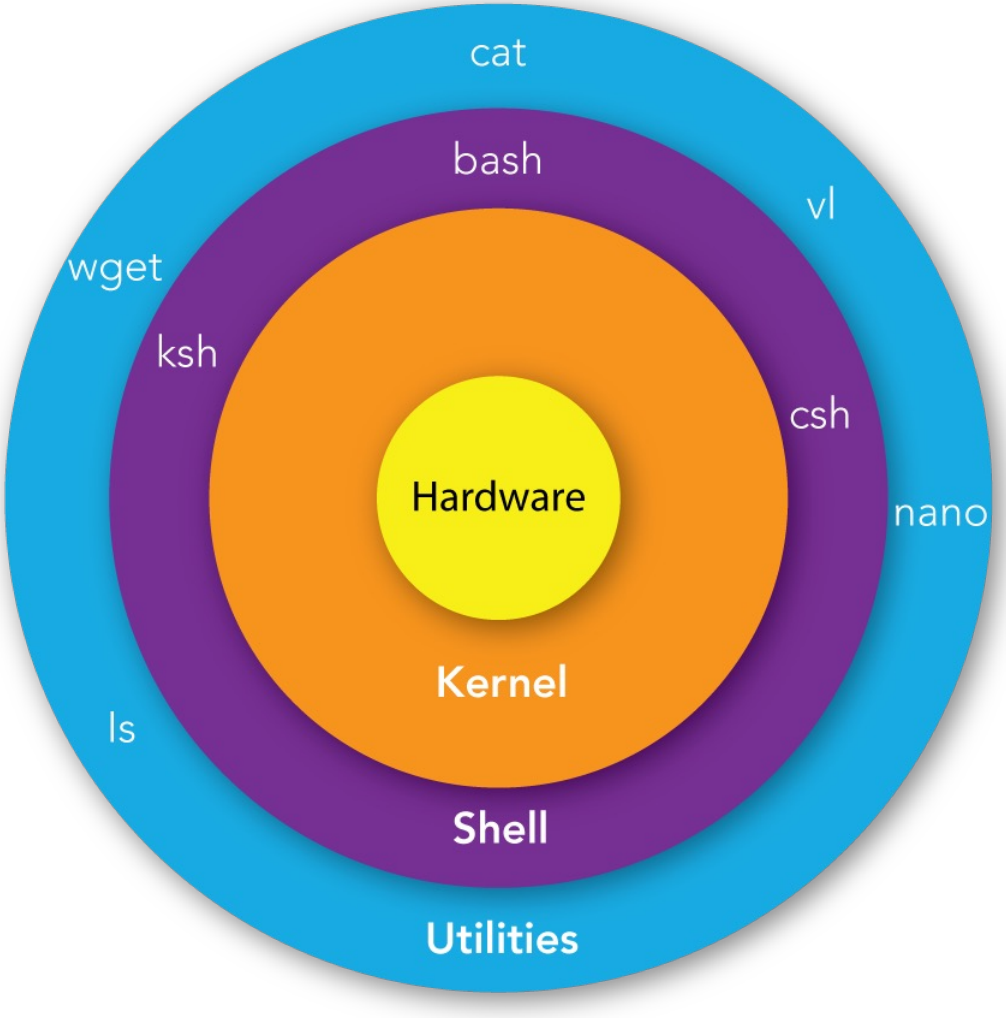
Introduction to Unix

Outline

1. File attributes and permissions
2. Basic Commands
3. Regular expressions
4. Interacting with the shell
5. Unix pipes
6. Job control
7. UNIX Environment variables
8. Text Editors
9. Shell scripting

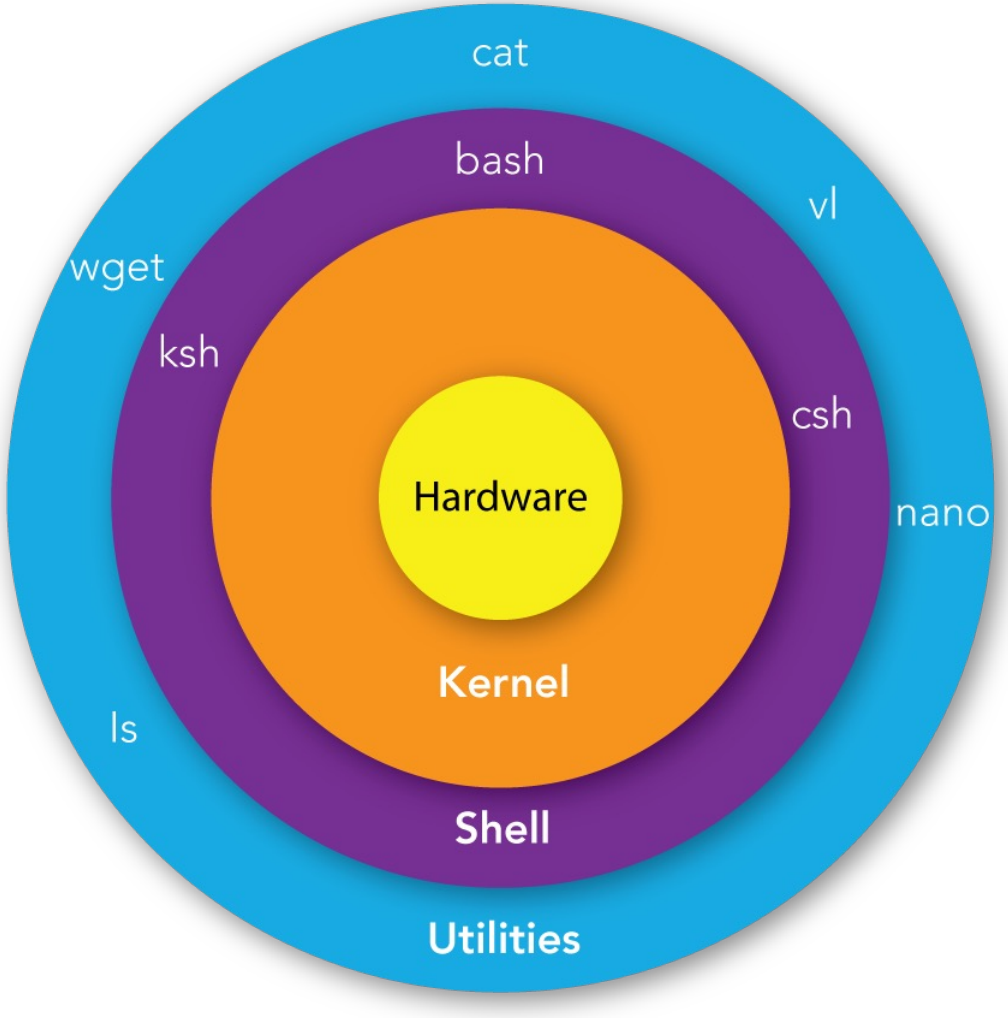
How does the OS work

- Everything in the OS is either a file or a process
- A process is an executing program identified by a unique PID (process identifier).
- A file is a collection of data. They are created by users using text editors, running compilers, etc.
- The kernel is responsible for organizing processes and interacting with files.



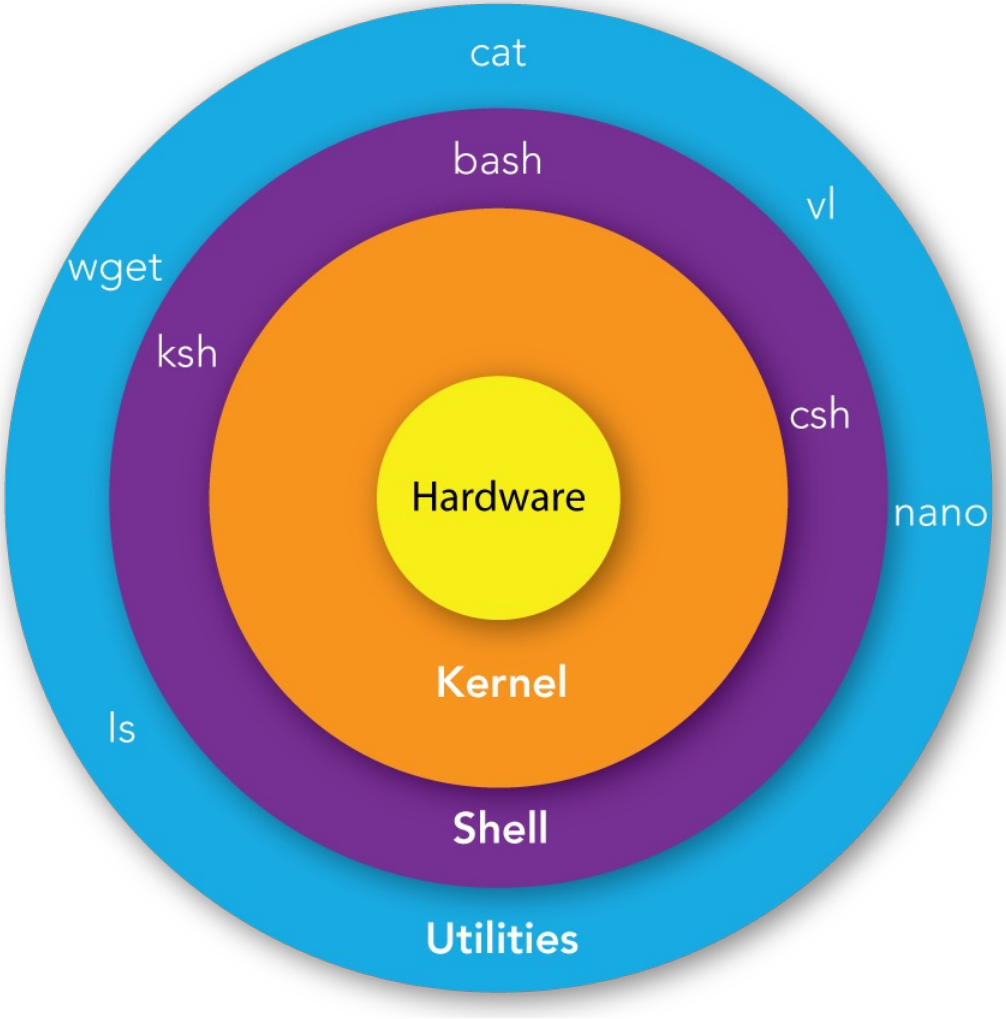
How does the OS work

- Shell is the OS's user interface.
  - Read command
  - Process command
  - Execute command
  - Display prompt



How does the OS work

- Example: suppose a user wants to remove a file
  - User has a command-line prompt
  - User types a command requesting the file removal (eg. rm myfile) in the shell
  - The shell searches the filesystem for the file containing the remove program (rm)
  - A new process is forked from the shell to run the command with an instruction to remove myfile
  - The process requests that the kernel, through system calls, delete the reference to myfile in the filesystem
  - When the rm process is complete, the shell then returns to the OS prompt indicating that it is waiting for further commands
  - The process ID (PID) originally assigned to the rm command is no longer active



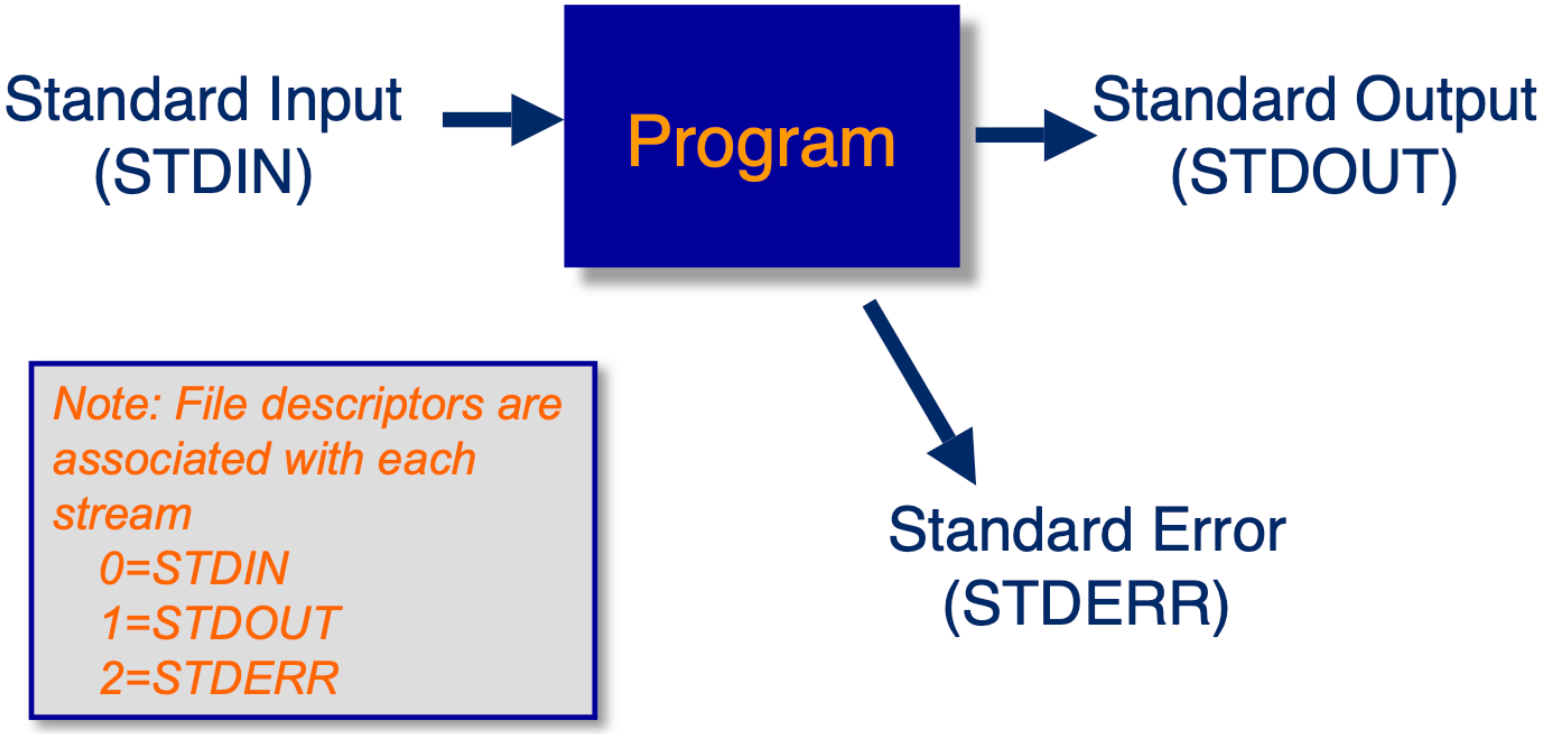
Shell flavors

- There are two main 'flavors' of shells:
  - Bourne created what is now known as the standard shell: "sh", or "bourne shell". Its syntax roughly resembles Pascal. Its derivatives include "ksh" ("korn shell") and now, the most widely used, "bash" ("bourne again shell")
  - One of the creators of the C language implemented a shell to have a "C-programming" like syntax. This is called "csh" or "C-shell". Today's most widely used form is the very popular "tcsh"
- Shells can run interactively or as a shell script

Interacting with the shell

- Typically, you type in the name of a program and some command line options
- The shell reads this line, finds the program and runs it, feeding it the options you specified
- The shell establishes 3 separate I/O streams:
  - Standard Input
  - Standard Output
  - Standard Error

Interacting with the shell



Interacting with the shell

- When a shell runs a program for you:
  - standard input is your keyboard
  - standard output is your screen or window
  - standard error is your screen or window
- If standard input is your keyboard, you can type stuff in that goes to a program
- To end the input you press Ctrl-D (^D) on a line by itself, this ends the input stream
- The shell is a program that reads from standard input
- Any idea what happens when you give the shell ^D?

Shell customization

- Each shell supports some customization
  - user prompt settings
  - environmental variable settings
  - aliases

- The customization takes place in startup files which are read by the shell when it starts up
  - Global files are read first – these are provided by the system administrators (e.g. /etc/profile)
  - Local files are then read in the user's HOME directory to allow for additional customization

Shell startup files

- sh,ksh:
- ~/.profile
- bash:
  ~/.bash_profile
  ~/.bash_login
- ~/.profile
- ~/.bashrc
- ~/.bash_logout
- csh:
- ~/.cshrc
- ~/.login
- ~/.logout
- tcsh:
- ~/.tshrc
- ~/.cshrc
- ~/.login
- ~/.logout

- These are hidden files in home directory
- .bashrc is executed for interactive ***non-login shells***
- .bash_profile or .bash_login or .profile is exectued for ***login shells*** (i.e. when you use password to enter shell)
- In login shell, only one of the above will be loaded.
- In Mac OS terminal by default runs a login shell every time.

Shell variables

- Shell variables can only contain letters (a-z A-Z), numbers (0-9), or the underscore (_).

```
_ALI
TOKEN_A
VAR_1
VAR_2
```

Correct.

```
2_VAR
-VARIABLE
VAR1-VAR2
VAR_A!
```

Wrong.

- Use = to define a variable (without spacing)
  e.g. machine_name=Kohn

- Access the variable value by $
  echo $machine_name

- Unsetting a variable directs the SHELL to remove the variable from the list of variables that it tracks.
  unset machine_name

Shell variables

Three types of variables:

- Local variables : a variable that is present within the current instance of shell command prompt. It is not available to programs started by the shell.

- Environmental variables : a variable available to any child process of the shell. Some functions need environmental variables to function correctly.

Define environmental variable : export var_name=var_value

It can be tedius to type 172.18.6.175 every time to enter Tai-Yi.

```
export TAIYI=172.18.6.175

ssh mae-liuj@$TAIYI
```

Shell variables

Three types of variables:

- Local variables : a variable that is present within the current instance of shell command prompt. It is not available to programs started by the shell.

- Environmental variables : a variable available to any child process of the shell. Some functions need environmental variables to function correctly.

Define environmental variable : export var_name=var_value

It can be tedius to remember the data directory on Tai-Yi.

```
export DATA=/data/mae-liuj

cd $DATA
```

Special environmental variable: PATH

- Each time you provide the shell a command to execute, it does the following:
  - Check to see if the command is a built-in shell command
  - If it is not a built-in command, the shell tries to find a program whose name matches the desired command

- How does the shell know where to look on the file system?

- The PATH variable tells the shell where to search for programs (non built-in commands)

- The PATH variable is a list of directories delimited by colons (:)
  - It defines a list and search order

```
-> env  list all environmental variables
TERM_PROGRAM=Apple_Terminal
SHELL=/bin/bash  shell type
TERM=xterm-256color
TMPDIR=/var/folders/w8/xz7sb8b54z1_zyrt75n8pyd80000gn/T/
TERM_PROGRAM_VERSION=433
TERM_SESSION_ID=217223CD-AE30-4FB1-9988-14DCB6E048D7
USER=juliu
SSH_AUTH_SOCK=/private/tmp/com.apple.launchd.F1SfIGgjKC/Listeners
PATH=/Users/juliu/lib/mpich-3.3.2/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
:/Library/TeX/texbin:/Library/Apple/usr/bin  we talked about this before
LaunchInstanceID=E8E6312D-FEA7-44D0-AB21-8EEDF6335D39
PWD=/Users/juliu
EDITOR=vim
XPC_FLAGS=0x0
XPC_SERVICE_NAME=0
SHLVL=1
HOME=/Users/juliu  home directory (~)
LOGNAME=juliu
VISUAL=vim
LC_CTYPE=en_US.UTF-8
SECURITYSESSIONID=186a9
FFLAGS=-w -fallow-argument-mismatch -O2
MACHINE_NAME=kolmogorov
_=/usr/bin/env
```

change your PATH variable

1. Go to your home directory
2. Display all files (including the hidder file)
3. Make sure the file .bashrc exists
4. Open .bashrc by a text editor
5. Add `export PATH=/Users/juliu/lib/mpich-3.3.2/bin:$PATH`
6. Save and exit the editor
7. Run source ~/.bashrc.

source is the command that loads the .bashrc configuration.

Wildcards for filename abbrevation

- When you type in a command line the shell treats some characters as special (*metacharacters)*
- These special characters make it easy to specify filenames
- The shell processes what you give it, using the special characters to replace your command line with one that includes a bunch of file names

see Chapter 7 of Unix in a nutshell

Wildcards for filename abbrevation

- “*” matches anything.
- If you give the shell “*” by itself (as a command line argument), the shell will remove the * and replace it with all the filenames in the current directory.
- “a*b” matches all files in the current directory that start with a and end with b.
- This looks like regular expressions but isn’t quite the same.

Wildcards for filename abbrevation

- The echo command prints out whatever you tell it:

```
$ echo hi
hi
```

- What will the following command do?

```
$ echo *
```

SHELL alias

- A shell alias is a shortcut to reference a command.
- Good for avoid typing long commands.
- Reduce keystronkes and guard correctness.

alias name=value

```
# User specific aliases and functions
alias ll='ls -lhrGS'
alias rm='rm -i'
alias mv='mv -i'
alias cdw='cd /work/mae-liuj'
alias cdd='cd /data/mae-liuj'
alias cds='cd /scratch'
alias rmsol='rm -rf SOL_* dot_SOL_*'
alias h5dump='/work/mae-liuj/lib/hdf5-1.8.16/bin/h5dump'
```