```
$
$
$
$ cat TOPICS.txt

* Introduction
* Navigating files and folders
* Working with files and folders
* Pipes and filters - the programming model
* Loops
* Writing shell scripts

$ whoami
```

Ge Baolai
SHARCNET
Western University
bge@sharcnet.ca

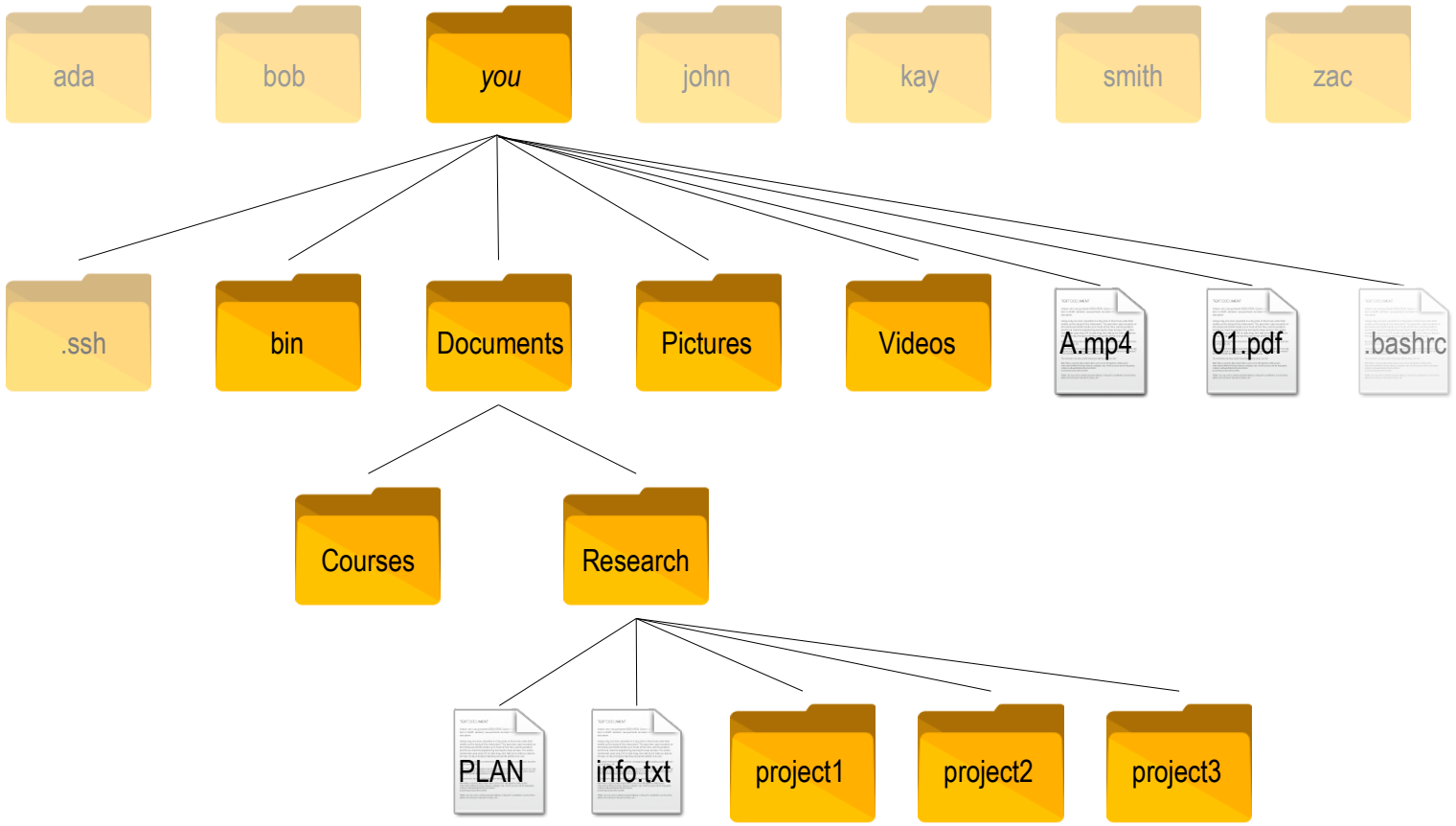
- In Windows, you move mouse point and click.
- In Linux, you open a text terminal – *a shell* – and type something in it. You will do everything from the prompt – command line, it's called command driven.
- It allows you to do things under the hood, and quick.
- Typically you will need to perform the following routine tasks in shell:
  - » Finding files and folders
  - » Editing files
  - » Compiling programs
  - » Running programs
  - » Running many programs at once, many, many times
  - » Processing results (data files), e.g. extracting a portion of data, etc.
  - » Copying, moving, deleting files and folders, etc.

all by hand…?

It's Linux

Navigating Files and Folders

Your files are stored on file systems. There are three file systems in SHARCNET you will care: /home/you, /work/you and /scratch/you.



It's Linux

Note the special files

- One dot "." means current folder
- Two dots ".." mean the "parent" folder or upper level foder
- A file or folder prefixed with a dot is "hidden" and not display with the command **ls**.
- To see all files including "hidden" files in the current folder, use ls command with option -a, e.g.

  ls -a

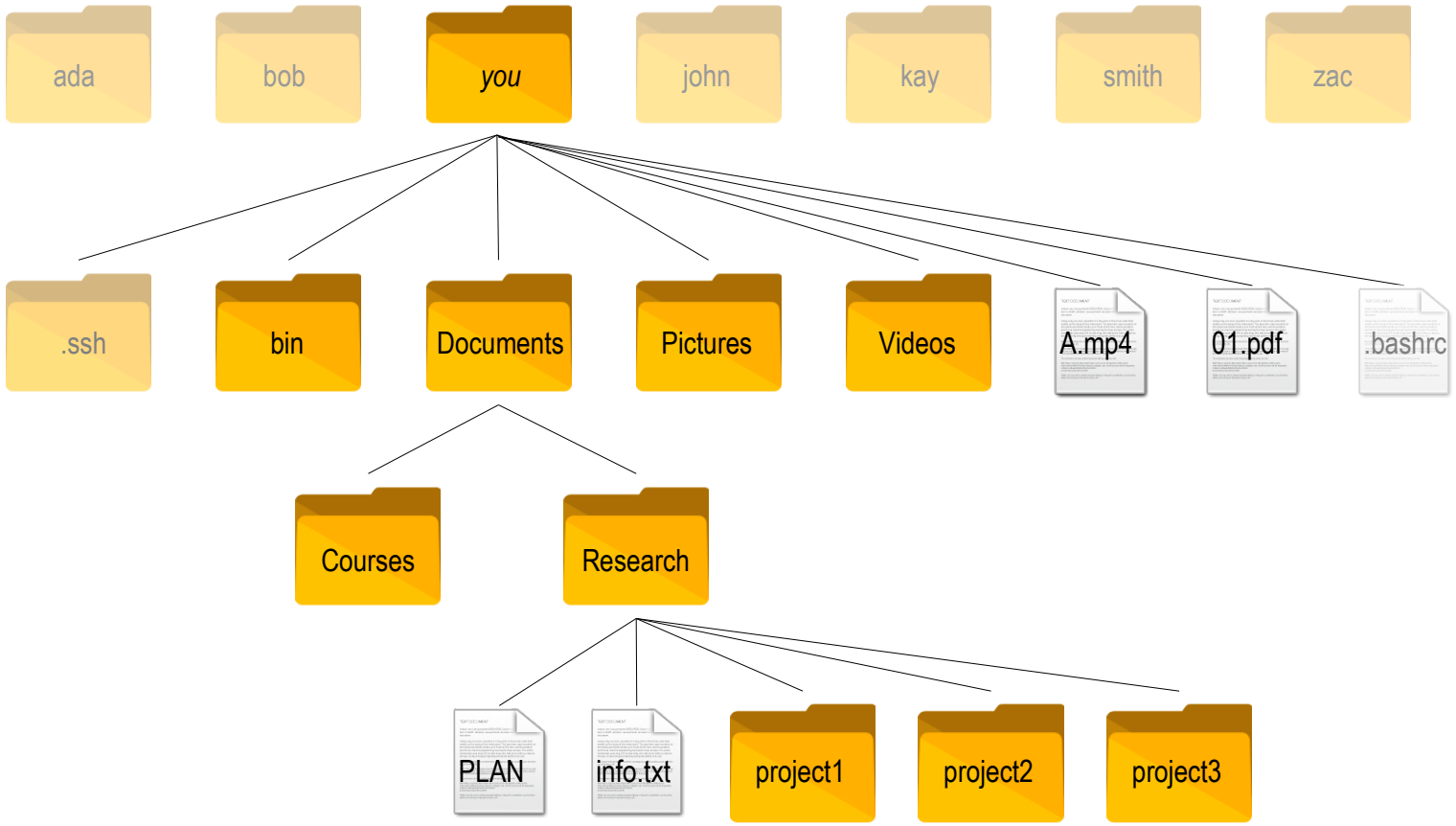
Commonly used shell commands

- **ls** – to list files and directories.
- **pwd** – to find the path of the current working directory.
- **cd** – change directory to.
- **whoami** – to find what my user name is.
- **file** – to find the type of a file.
- **find** – to serachto find a file by name.
- **locate** – to find a file by name.
- **stat** – to find the existence of a file/folder.
- **grep** – to search files by content matching certain pattern.
- **man** – to see the manual for a command, look for See Also section to see more related commands – a way of getting to know more commands.

It's Linux

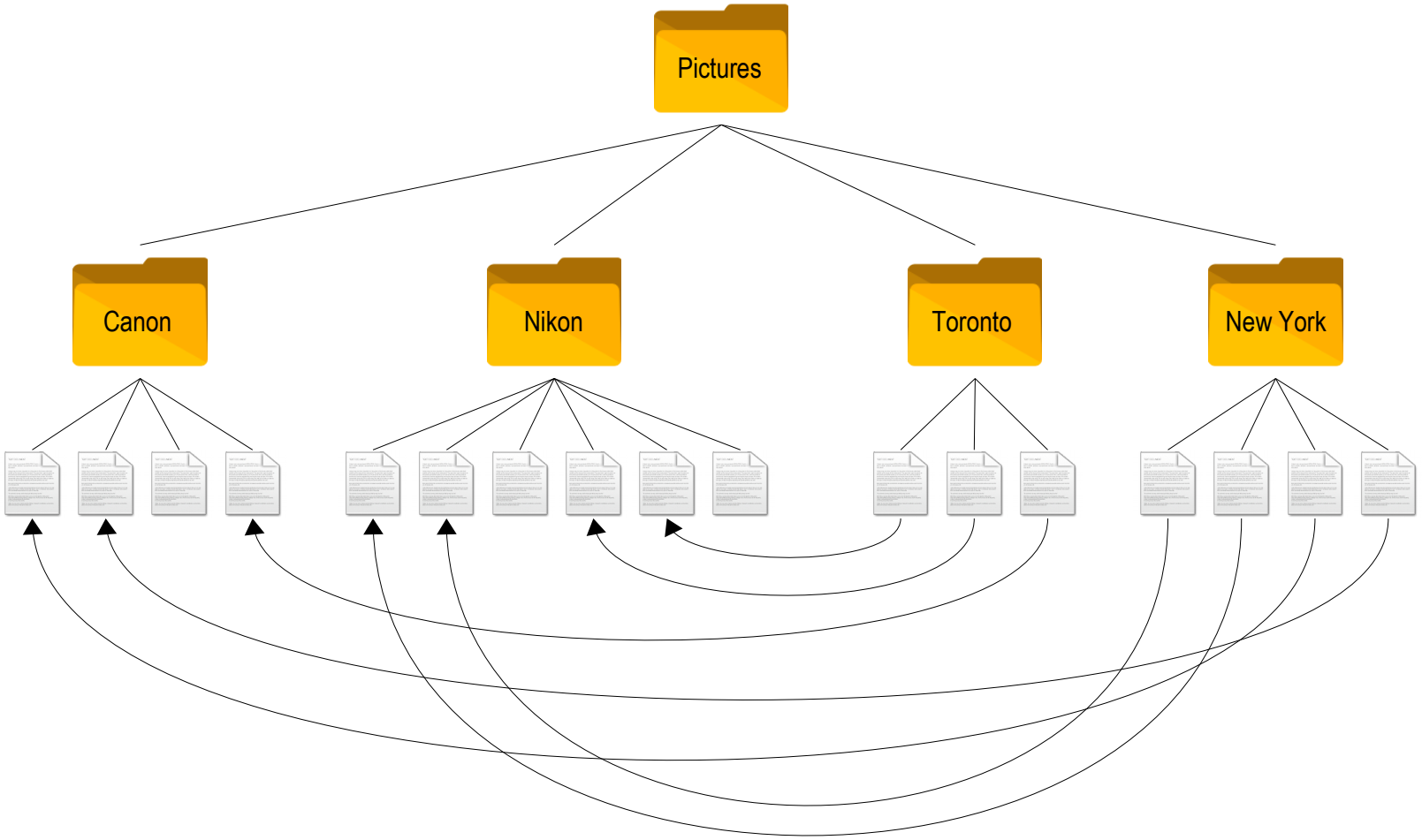
Working with Files and Folders

Your files are stored on file systems. There are three file systems in SHARCNET you will care: /home/you, /work/you and /scratch/you.



It's Linux

Use symbolic links to avoid duplicating files: we save original files, make "copies" by creating links to the original copies, rather than duplicating the content.



It's Linux

Example: Creating symbolic links

```
cd Pictures
# When creating a folder name containing a space
# use back slash followed by a space
mkdir Toronto New\ York

cd Toronto
ln -s ../Canon/2015_05_01/* .

cd ../New\ York
ln -s ../Canon/2015_01_01/* .
ln -s ../Nikon/2015_01_01/* .
```

It's Linux

Commonly used shell commands

- **ls** – to list files and directories.
- **cat** – to concatenate files and print on the standard output.
- **less, more** – to show a file.
- **cmp** – to find two files byte by byte.
- **diff** – to compare tow files line by line.
- **cp** – to copy a file/folder.
- **mv** – to move/rename file/folder.
- **rm** – to remove a file/folder.
- **mkdir** – to create a folder
- **rmdir** – to remove a folder.
- **chmod** – to change the access permission.
- **vi**, **nano**, **emacs**, **gedit**, etc. – create and editing files.

It's Linux

Pipes and Filters

We will see three (or more) common tasks accomplished by using shell commands:

- Counting lines, words and bytes of a file.
- Extracting a column from a well formatted tabulated data file.
- Extracting lines from certain range from a file.
- Saving the output of a command into a file.
- Automating simple tasks by combining commands.

It's Linux

Example: Change upper case letters to lower ones in a text file:

```
2012-11-05,DEER
2012-11-05,RABBIT
2012-11-05,RACCOON
2012-11-06,RABBIT
2012-11-06,DEER
2012-11-06,FOX
2012-11-07,RABBIT
2012-11-07,BEAR
```

Before

```
tr [A-Z] [a-z] < ANIMALS.txt
```

```
2012-11-05,deer
2012-11-05,rabbit
2012-11-05,raccoon
2012-11-06,rabbit
2012-11-06,deer
2012-11-06,fox
2012-11-07,rabbit
2012-11-07,bear
```

After

It's Linux

Example: Extract cities in Ontario from **cities.txt** and sort by population

```
Toronto,Ontario,5583064,1
Montréal,Quebec,3824221,2
Vancouver,British Columbia,2313328,3
Ottawa,Ontario,1236324,4
Calgary,Alberta,1214839,5
Edmonton,Alberta,1159869,6
Québec City,Quebec,765706,7
Winnipeg,Manitoba,730018,8
Hamilton,Ontario,721053,9
Kitchener-Cambridge-Waterloo,Ontario,477160,10
London,Ontario,474786,11
St. Catharines-Niagara,Ontario,392184,12
Halifax,Nova Scotia,390328,13
Oshawa,Ontario,356177,14
Victoria,British Columbia,344615,15
Windsor,Ontario,319246,16
Saskatoon,Saskatchewan,260600,17
Regina,Saskatchewan,210556,18
```

It's Linux

Example: Extract cities in Ontario (cont'd)

We now put the commands all together to accomplish this task

```
grep -i ontario cities.txt | cut -d ',' -f 1,3 | sort -k 2 -t ',' -n
```

The first command with **grep** outputs only the lines that contains Ontario. It then pipes the output as the input to the second command **cut**.

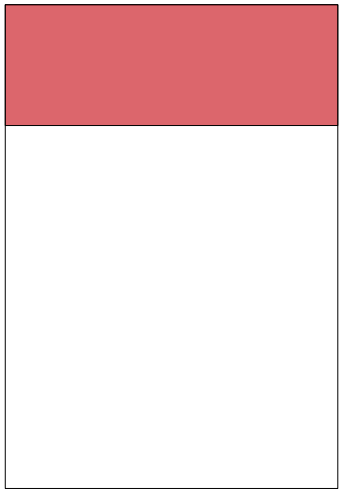
The command cut filters out the first and third fields and outputs the city and population columns (to the standard out).

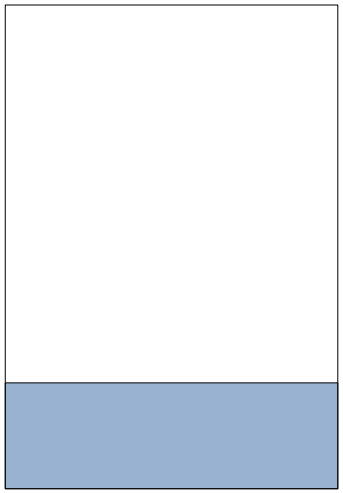
The third command **sort** takes output from the previous command cut ans input and sort by the second field (with option -k 2) in a reverse order (with option -n). The option -t ',' specifies the field separator.

It's Linux

Example: Extracting first and last number of lines from a file.

head -n *m*

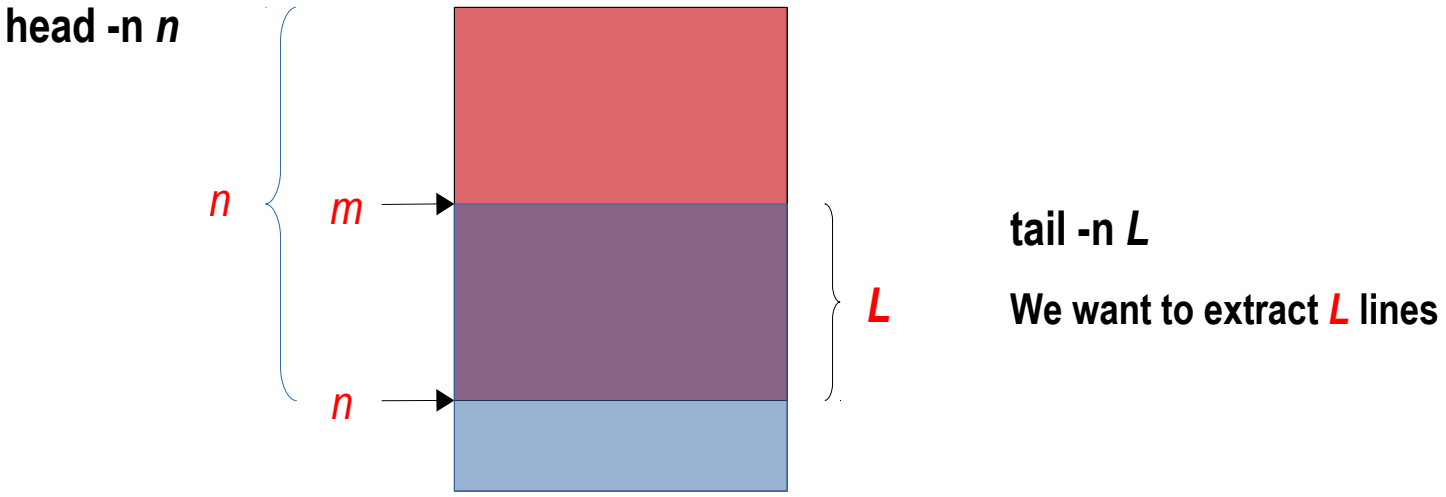




tail -n *n*

It's Linux

Example: Extracting lines from certain range from a file using combined commands head and tail.



$L = n - m$

head -n *n* … tail -n *L* ?

It's Linux

Commonly used shell commands

- **wc** – to count lines, words and bytes of a file.
- **cut** – to extract column(s) from a file by delimiters.
- **tr** – to translate or delete characters.
- **sort** – to sort a file by rules.
- **uniq** – to report or omit repeated lines.
- **head** – to print the first N lines of a file.
- **tail** – to print the last N lines of a file.

It's Linux

Loops

Common tasks

- Repeating tasks by looping over control variables, e.g n=1,2...
- Repeating tasks by looping over list elements.

It's Linux

Example of loops:

```
# For loops
for ((i=0;i<10;i++)); do
    echo $i
done

list="a b c d"
for e in $list; do
    echo $e
done

for f in *; do
    echo $f
done

# While loops
n=10
i=$n
while [ $i -gt 0 ]; do
    echo $i;
    i=$((i-1));
done
```

Example: Read a file containing multiple lines of records in a loop and process one at a time.

```
input="animal.txt"
BAKIFS=$IFS
IFS=$(echo -en "\n\b")
cat $input | while read line; do
    animal=`echo $line | cut -d',' -f2`
    echo $animal
done
IFS=$BAKIFS
```

It's Linux

Example: Organizing phone photos into folders by shooting date.

Before

```
[bge@parrot:~/Documents/teaching/bash/exercises/phone] dir
IMG_20150101_000120.jpg  IMG_20150206_070150.jpg  IMG_20150410_064001.jpg
IMG_20150101_000237.jpg  IMG_20150206_070422.jpg  IMG_20150410_064011.jpg
IMG_20150101_000251.jpg  IMG_20150209_160518.jpg  IMG_20150608_172050.jpg
IMG_20150113_071752.jpg  IMG_20150209_160525.jpg  IMG_20150608_172109.jpg
IMG_20150113_113728.jpg  IMG_20150221_143202.jpg  IMG_20150608_181324.jpg
IMG_20150120_071809.jpg  IMG_20150227_174437.jpg  IMG_20150608_185905.jpg
IMG_20150120_071815.jpg  IMG_20150306_074007.jpg  IMG_20150724_084733.jpg
IMG_20150128_071353.jpg  IMG_20150306_144429.jpg  IMG_20150819_145203.jpg
IMG_20150128_071517.jpg  IMG_20150306_144621.jpg  IMG_20150819_145218.jpg
...
```

After

```
[bge@parrot:~/Documents/teaching/bash/exercises/phone] dir
2015_01_01  2015_01_28  2015_02_21  2015_03_08  2015_07_24
2015_01_13  2015_02_06  2015_02_27  2015_04_10  2015_08_19
2015_01_20  2015_02_09  2015_03_06  2015_06_08
```

It's Linux

Example: Organizing phone photos into folders by shooting date (cont'd).

```
for f in IMG*.jpg; do
    file_date=${f:4:8}   # Extract the date string
    yy=${file_date:0:4}  # Extract the year part
    mm=${file_date:4:2}  # Extract the month part
    dd=${file_date:6:2}  # Extract the date part
    d=${yy}_${mm}_${dd}  # Rebuild the date string
    mkdir $d
    echo "Moving $f to $d"
    mv $f $d
done
```

It's Linux

Shell Scripts

Putting tasks together in a file to execute

- Putting commands together – simplest case.
- Putting together flow control with shell programming constructs – if-else, loops, etc.

It's Linux

SHARCNET™ Western

Example: Organizing phone photos into folders by shooting date.

```
#!/bin/bash
# 'photosort' - Reorganize photos by shooting dates
for f in IMG*.jpg; do
    # Get the shooting date from file name
    file_date=${f:4:8}  # Extract the date string
    yy=${file_date:0:4} # Extract the year part
    mm=${file_date:4:2} # Extract the month part
    dd=${file_date:6:2} # Extract the date part
    d=${yy}_${mm}_${dd} # Rebuild the date string
    # Create the folder if not present
    if [ ! -d $d ]; then
        mkdir $d
    fi
    echo "Moving $f to $d"
    mv $f $d
done
```

It's Linux

Example: Organizing phone photos into folders by shooting date (cont'd).

```
# Change the permission mode of the file, make the script executable
chmod photosort

# Go to the folder containing the images
cd Pictures/Huawei

# Run the script (“./” means the script is in the current folder)
./photoshort
```