Introduction to Unix for CRG Postdocs – 02/14/2011 - PRBB

1) Open an xterm
2) Basic syntax of a command:
   → `command option1 option1value option2 option2value argument`
3) To get the manual of a command:
   → `man command`

I. Commands to navigate on the system

- `pwd`:
  - outputs your working directory = where you are.
- `ls`:
  - lists the files and directories present in your working directory.
  - `ls -lrt`: long listing and by order of modification date
- `cd`:
  - `cd dir` = goes to directory `dir` (`dir` is either absolute or relative).
  - `cd` or `cd ~` = goes back to your home directory.
- `mkdir`:
  - `mkdir dir` = creates the empty directory `dir` in the place where you are.
- `rm`:
  - `rm file` = removes file
  - `rm -r dir` = removes directory dir (rmdir dir only works for empty dir)
- `cp`:
  - `cp file dir` = copies `file` in `dir.`
  - `cp file1 file2` = creates `file2` as a copy of `file1.`
  - `cp -r dir1 dir2` = copies `dir1` in `dir2.`
- `mv`:
  - `mv file1 file2` = renames `file1 file2`
  - `mv file dir` = moves `file` in `dir`
  - `mv dir1 dir2` = moves `dir1` in `dir2`
- `date`:
  - prints out the current date and time.
- `file`:
  - `file file1` = determine file type for `file1`
- `tree`:
  - lists contents of directories in a tree-like format.
- `up arrow` (on the prompt):
  - gets the commands you previously typed in.
- `tab` when typing in a command or a file:
  - `auto-completes` any command or file you type in.
- `control r` (on the prompt) + `keyword` :
  - finds any previous command containing `keyword.`

- `mouse-recording` text to copy/paste:
  - `1 click` = records the character.
  - `2 clicks` = records the word.
  - `3 clicks` = records the line.
- `find`:
  - searches for files in a directory hierarchy.
  - `find ~ -name file` = finds file in your home directory and anywhere down it.
  - `find . -name file` = finds file where you are and anywhere down it.
- `emacs`:
  - text editor widely used under unix, using buffers and windows, convenient for programming.
  - `control x control f file:` to open `file` in current buffer.
  - `control x control s:` to save current file.
  - `control s word:` to search for `word` in current buffer`.`
  - `esc x % word1 enter word2 enter:` to replace `word1` by `word2` everywhere in current buffer.
  - `esc g line_no: to` go directly to line `line_no` of current buffer.
  - `control x control c`: to quit emacs.
  - `mouse-record text + control y`: paste `text`.
  - `mouse-record text + control w`: cut `text`.

II. Basic commands on files

- `more/less`:
  - `more file`: pages through text in `file` one screenful at a time.
- `head`:
  - outputs the first part of files (10 first lines by default).
  - `head -n 100 file` : outputs the 100 first lines of `file`.
- `tail`:
  - outputs the last part of files (10 last lines by default).
  - `tail -n 50 file`: outputs the last 50 lines of `file`.
- `wc`:
  - prints number of lines, words, and characters present in a file.
  - `wc -l file`: prints the number of lines of `file`.
- `grep`:
  - prints lines matching a pattern.
  - `grep (-w) word file`: finds all the lines in `file` containing (exactly) the word `word`.
  - `grep -f file1 file2`: finds all the lines in `file2` containing the patterns indicated in `file1`.
- `diff`:
  - finds differences between two files.
  - `diff file1 file2`
- `echo`:
  - displays a line of text.
  - `echo hello`: prints the “hello” string on the screen.
- `Standard output` and `standard error`:
  - by default all the unix command print their result on the standard output (= the

screen), and their error in the standard error (= the screen).
    - It is possible to redirect a command result or/and error in a file using:
        - `command > outfile 2> errorfile`
        - `echo hello > file`: prints the “hello” string in file.
- `cat`:
    - concatenates files and print on the standard output.
    - `cat file1 file2 > file3`: appends `file2` at the end of `file1` and outputs the result in `file3`.
- `paste`:
    - merges lines of files.
    - `paste file1 file2 > file3`: to each line of `file1` appends the corresponding line from `file2` and outputs the result in `file3`.
- `sort`:
    - sorts lines of text files in a very efficient way (quick).
    - `sort file1 > file2`: sorts `file1` alphabetically and outputs the result in `file2`.
    - `sort –n file`: sorts `file` numerically.
    - `sort –k3,3nr file`: sorts `file` according to its column no 3 and in a numerically reverse way.
- `cut:`
    - removes sections from each line of files (default separator is tab).
    - `cut –f 1,3 file:` outputs columns 1 and 3 of `file`.
    - `cut –f 1–3 file`: outputs column 1 to 3 of `file`.
- | (called pipe):
    - `command1 | command2`: redirects the output of `command1` into the input of `command2`, thus avoiding writing intermediate file containing the result of `command1`.
- `uniq`:
    - reports or f filters adjacent matching lines from input writing to output.
    - `uniq (–c) file`: matching lines of `file` are merged to the first occurrence (and lines are prefixed by the number of occurrences).

III. Awk = pattern scanning and processing language.

- Powerful but easy to learn scripting language for working on structured files seen as lists of strings.
- Principle: execute a given action on each line of a given input file.
- Syntax:
    - `awk '{action}' inputfile > outputfile`
- Note that it can also execute a given action before or/and a given action after reading the input file:
    - `awk 'BEGIN{begactions} {actions} END{endactions}' inputfile > outputfile`
    - When the program (= what is between the simple quotes) grows, it is always better (less error prone and more confortable since common editors put colors and correctly indent the program) to put it in a file (`script.awk`) and call it as follows:
    - `awk –f script.awk inputfile > outputfile`
- Each line is seen as a set of strings separated by separators that are by default any combination of tabs and spaces (this can be changed). For the current input line:

    - the value / string of field no 1 is `$1`
    - the value / string of field no 2 is `$2`
    - …
- `$0` means the entire current input line.
- Special variables are:
    - `NR`: current line number
    - `NF`: total number of fields in current line → value / string corresponding to last field of a line is `$NF`
- Assignment of variable `var` to value `val`:
    - `var=val;`
    - `val` can be an integer, a float, a string, a character.
- Test of variable `var` to value `val`:
    - `var==val`
    - `awk 'NR==5' infile`
        - will print out line no 5 of `infile`
        - is a shortcut of `awk '{if(NR==5){print $0;}}' infile`
    - `can be put in a variable using var=`awk ...``
- `Access to subsets of a file path`
    - `dirname $f : directory of file $f`
    - `basename $f : name of file $f (without directory)`
    - `${f%.ext} : removes .ext at the end of $f`
    - `${f#sth} : removes sth at the beginning of $f`
- `For` and `while` loops as in other lamguages
    - `for(i=1; i<=n; i++){actions}`
    - `while(i<=n){actions}`

IV. Bash = sh-compatible command language interpreter that executes commands read from the standard input or from a file.

- Can be used on the command line or via a file. In this last case we have a stand-alone program to which we can pass parameters and that can be reused.
- Very powerful in combination with awk and other linux commands to manipulate and extract information from multiple files on a system.
- Assign variable `var` the value `val`:
    - `var=val;`
- Access to value of variable `var`:
    - `$var`
- Do an action on several files:
    - `ls * | while read f; do actions $f; done`
- Do an action on each line of a file:
    - `cat file | while read a b c; do actions $a $b $c; done`
- Do an action on files indexed by an integer `i`:
    - `for i in $(seq 1 20); do actions file_$i.txt; done`
- Do an action if a condition `cond` is fullfilled:
    - `if [ cond ]; then actions; fi`

`Comparisons:`

-eq	equal to
-ne	not equal to
-lt	less than
-le	less than or equal to
-gt	greater than
-ge	greater than or equal to

File Operations:

-s	file exists and is not empty
-f	file exists and is not a directory
-d	directory exists
-x	file is executable
-w	file is writable
-r	file is readable

An example of script with if condition:

```
#!/bin/sh
# This is some secure program that uses security.

VALID_PASSWORD="secret" #this is our password.

echo "Please enter the password:"
read PASSWORD

if [ "$PASSWORD" == "$VALID_PASSWORD" ]; then
        echo "You have access!"
else
        echo "ACCESS DENIED!"
fi
```

Another example of script with if condition:

```
#!/bin/sh
# Prompt for a user name...
echo "Please enter your age:"
read AGE

if [ "$AGE" -lt 20 ] || [ "$AGE" -ge 50 ]; then
        echo "Sorry, you are out of the age range."
elif [ "$AGE" -ge 20 ] && [ "$AGE" -lt 30 ]; then
        echo "You are in your 20s"
elif [ "$AGE" -ge 30 ] && [ "$AGE" -lt 40 ]; then
        echo "You are in your 30s"
elif [ "$AGE" -ge 40 ] && [ "$AGE" -lt 50 ]; then
        echo "You are in your 40s"
fi
```