Shell Environments

The Shell Environment

- Shell environment
  - Consists of a set of variables with values.
  - These values are important information for the shell and the programs run from the shell.
    - Example: PATH determines where the shell looks for the file corresponding to your command.
    - Example: SHELL indicates what kind of shell you are using.
  - You can define new variables and change the values of the variables.

Shell Variables (1)

- Shell variables are used by putting a $ in front of their names
  - e.g. echo $HOME
- Many are defined in .bash_profile and .bashrc
- Two kinds of shell variables:
  - Environment variables
    - available in the current shell and the programs invoked from the shell
  - Regular shell variables
    - not available in programs invoked from this shell

Shell Variables (2)

- In bash, sh, and ksh, regular variables are defined in the following way:

  varname=varvalue

- In bash, sh, and ksh, environment variables are called “exported variables” and are defined in the following way:

  MYENVVAR="env var”

  export MYENVVAR

  or export MYENVVAR=“env var”

- Clear a variable: unset varname;

Shell Variables (3)

- Example:

```
compute[3] > export MYENVVAR="Unix is easy"
compute[4] > myregvar="Windows is easy"
compute[5] > bash
compute[6] > echo $MYENVVAR
Unix is easy
compute[7] > echo $myregvar
bash: myregvar: unbound variable
compute[8] >
```

Or

```
compute[7] > echo $myregvar

compute[8] >
```

Here we enter a new shell…

If we have set –u in .bashrc

Shell Variables (4)

- In csh and tcsh, setting regular variables:
  - set varname=varvalue

```
compute[4] > set myvar="reg var"
compute[5] > echo $myvar
reg var
```

- Clearing out regular variables:

```
compute[4] > unset myvar
compute[5] > echo $myvar
myvar: undefined variable
```

- Setting environment variables:

```
compute[1] > setenv MYENVVAR  "env var "
compute[2] > unsetenv MYENVVAR
```

  - *No "=" sign here!*

Shell Variables (5)

- Common shell variables:
  - SHELL: the name of the shell being used
  - PATH: where to find executables to execute
  - LANG: the locale you are using
  - LIBRARY_PATH: where libraries for executables are found at run time
  - USER: the user name of the user logged in
  - HOME: the user's home directory
  - TERM: the kind of terminal the user is using
  - DISPLAY: where X program windows are shown
  - HOSTNAME: the name of the host logged on to

More on Unix Quoting

- Single Quotes '...'
  - Stop variable expansion ($HOME, etc.)

```
compute[1] > echo "Welcome $HOME"
Welcome /home/kzhang
compute[2] > echo 'Welcome $HOME'
Welcome $HOME
```

- Back Quotes `…`
  - Replace the quotes with the results of the execution of the command.
  - E.g.

```
compute[3] > PS1=`hostname`
```

  - More standard way

```
compute[4] > PS1=$(hostname)
```

The Search Path

- How does Unix find commands to execute?
  - If you specify a pathname, the shell looks into that path for the executable.
  - If you specify a filename, (without / in the name), the shell looks for it in the search path.
  - There is a variable PATH or path (in csh and tcsh)

    compute[1] > echo $PATH

    /usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/kzhang/bin:.
- The shell does not look for executables in your current directory unless:
  - You specify it explicitly, e.g. ./a.out
  - . is specified in the path variable

Selecting Different Versions of a Command

- There may be multiple versions of the same command in your search path.

```
compute[1] > whereis ps
ps: /usr/bin/ps /usr/ucb/ps
```

- The shell searches in each directory of the $PATH in left to right order and executes the first version.

```
compute[2]> which ps
/usr/bin/ps
compute[3]> /usr/ucb/ps
```

Shell Startup

- When bash is executed, it runs certain configuration files:
  - .bash_profile run once when you log in
    - Contains one-time things like terminal setup.
  - .bashrc run each time another bash process runs
    - Sets lots of variables and aliases.
- Other shells such as tcsh use different files
- Only modify the lines that you fully understand!

The alias Command

- alias format:
  - alias alias-name='real-command'
    - alias-name is one word
    - real-command can have spaces in it
- Any reference to alias-name invokes real-command.
- Examples:
  - alias rm='rm –i'
  - alias cp='cp –i'
  - alias mv='mv –i'
  - alias ls='/usr/bin/ls –CF'
    - This shows us the /, *, @ after file names using ls.
- Put aliases in your .bashrc file to set them up whenever you log in to the system!

Command History (1)

- compute[9] > history

```
    1   emacs
    2   ls -l .cshrc
    3   cp .cshrc .cshrc2
    4   emacs .cshrc
    5   ps
    6   pwd
    7   cd ..
    8   pine
    9   history
```

Command History (2)

- You can rerun a command line in the history
  - !! reruns last shell command
  - !str reruns the latest command beginning with str
  - !n (where n is a number) reruns command number n in the history list
- bash allows you to use arrow keys to wander the history list easily.
- The length of the history list is determined by the variable HISTSIZE, likely set in your .bashrc file.

  HISTSIZE=400
- The variable HISTFILESIZE determines how much history to save in the file named .bash_history for your next session.

Command and Filename Completion

- In bash and tcsh, you can let the shell complete a long command name by:
  - Typing a prefix of the command.
  - Hitting the TAB key.
  - The shell will fill in the rest for you, if possible.
- bash and tcsh also complete file names:
  - Type first part of file name.
  - Hit the TAB key.
  - The shell will complete the rest, if possible.
- Difference:
  - First word: command completion.
  - Other words: file name completion.

Some Useful Commands

- bash and tcsh (for tcsh set autolist)

  tab completion will show a list of possibilities when the completion choice is ambiguous

- export PATH=$PATH:$HOME/bin:.

  if you want to add ~/bin and . to your PATH

- export TERM=xterm

  if your terminal is not working properly, i.e. some system does not know xterm-256color

- printenv or env

  show the current values of all your environment variables