# CSE 303
# Concepts and Tools for Software Development

Magdalena Balazinska
Winter 2010
Lecture 2 – Filesystem, processes, users, and command line

## Class Mailing List and HW1

- You should have received an email from me yesterday about assignment 1
  - If you did not receive this email, let me know
- After today, you will know enough to start HW1
  - You will need next lecture to finish it

## Where We Are

- It's like we started over using the computer from scratch
- And all we can do is run dinky programs at the command-line
- But we are learning
  - A model: filesystem, processes, users
  - A powerful way to control it: the shell
- Last time: filesystem and shell basics

# Some Useful Commands

- Navigating directory structure: `cd,pwd,ls`
  - Relative path: `cd ../cse303`
  - Absolute path: `cd /home/username/cse303`
- Manipulating files: `mv, cp, rm`
- Manipulating directories:
  - `mkdir, rmdir, cp -r, rm -rf`
- Viewing file content: `cat,head,tail,less`
- Changing permissions: `chmod`
  - Example: `chmod -R go-rw .`

- Linux Pocket Guide p.37 and later!

## Outline for Today

- The rest of the model
  - Users
  - Programs and processes
- The power of the shell (just the beginning)
  - Special characters: file metacharacters

## Users

- One filesystem and one operating system
- But many users
  - home directory, permissions, `whoami`, `quota`
  - change permissions with `chmod`
    - You can use it to make your homework unreadable by others ;-)
  - one “superuser”: `root` (administers machine)

- Different users can access the same physical machine at the same time (e.g., four attu machines)

# At login

- `/etc/passwd` guides the *login* program
  - Verifies user name and password
  - Sets some environment variables: HOME, PATH
  - Launches the appropriate shell
  - The shell then takes over with startup scripts
    - `/etc/profile`
    - `~/.bash_profile`

- But passwords are in `/etc/shadow`
  - Why? Hint: compare permissions on these files
  - If we made /etc/passwd NOT world readable, `ls` would display user IDs instead of names and other such problems

## Tailoring Your Shell Behavior

- Create and edit files `.bashrc` and `.bash_profile`
  - In hw1, we give you a `.bash_profile` that runs `.bashrc`, so you only need to modify the latter
  - Those files must be placed directly in your home directory

- Use Linux Pocket Guide (LPG) to lookup difference between `.bashrc` and `.bash_profile`
  - `.bash_profile` executes when logging in
  - `.bashrc` executes when opening a new shell

## Tailoring Your Shell Behavior

- You can put almost anything you want in these files
  - Common use is to define aliases and environment variables
  - To declare a variable and assign a value:
    - `MYVAR=value`
  - To make variable available to subshells and programs
    - `export MYVAR`
  - Short hand: `export MYVAR=value`
- Two examples
  - `export CLASSPATH=~/lecture2/hello.jar`
  - `export CLASSPATH=$CLASSPATH:~/lecture2/hello.jar`
- We will learn more about shell variables in next lecture

## Processes

- A running program is a process
- An application may run many processes
- The shell runs a program by
  - "Launching a process"
  - Waiting for the process to finish
  - Giving the prompt back
- A running shell is just a process that kills itself when interpreting the `exit` command
- GUIs are just a type of application

## Program Options

- Most programs have options
- Single-letter preceded by a single hyphen

```
rm -r -f *
rm -rf *
```

- Or long options preceded by 2 (or 1) hyphens

```
ls --color
```

- Some commands support both

```
grep -c cat *.txt
grep --count cat *.txt
```

## Discovering Available Options

- Program man takes a program name and displays the manual page or manpage
- Standard option `-help`
  - Prints usage and exits
  - Often programs print usage when given bad options
- Resources on the Web
  - Google is your friend

## Controlling Processes

- Possible to run a program in the background
  - `C-Z, fg, bg, &`
- Viewing processes and killing them

```
jobs, ps, top, kill, ^C
```

## Summary of System Model

- Filesystem: tree of directories and files
- Users: home directory, permissions
- Processes that
  - Perform some useful work
  - Perform Input/Output (I/O)
  - Interact with devices: monitor, keyboard, network
  - Launch other processes
  - Create and modify files or directories
- The operating system manages all these

# The Shell: What We Know So Far

- Program that interprets commands and initiates their execution
- Additionally, the shell has a state
  - Current working directory
  - Current user, her home directory, etc.
- Builtins: commands provided by the shell
  - `cd, exit, echo, source, alias`
  - Give directives to the shell
  - Change the state of the shell

## File Metacharacters

- The shell provides powerful features to make the user’s life easier: i.e., speed-up the user’s work
- One such feature is to provide file metacharacters
- The shell performs various expansions and substitutions **before** invoking a program
- Example: `ls -l *.txt`
- Why file metacharacters?
  - On the command line: save typing
  - Inside scripts: flexibility (ex: email all pictures)

# Expansions

- Brace expansion
  - Example: `mkdir hw1/{old,new,test}`
  - Creates: `hw1/old, hw1/new, hw1/test`
- Tilde expansion (expansion of ~ character)
  - Home directory of user bob: `~bob`
  - Current user's home directory: `~`
- Filename expansion: `*, ?, [`
  - Replace pattern with list of matching file names

## Pattern Matching

- Any string, including null string: `*`
- Any single character: `?`
- Any character from set: `[ ]`
  - Example `[abc]` or `[a-c]`
- Any character not in set: `[!abc]` `[^abc]`
- **Special case**: “`.`” at beginning of a file name
- Examples:
  - `mv mytaxes*19* very-old`
  - `mv mytaxes*200[0-4]* old`

# Special Characters

How to use them without special meaning?

- Escape: `\x` takes following character, `x`, literally
- Single quotes: `'xxx'` take everything literally
- Double quotes: `“xxx”` take everything literally except `$`, ` `` ` (for command subst.), and `\` if followed by special character

- Rules on what to escape or quote are arcane
  - When in doubt, just give it a try

## Quoting and Escaping Examples

Directory contains three files: `a.txt, a*.txt, a?*.txt`

```
> ls a*.txt
> a?*.txt  a.txt  a*.txt
> ls a\*.txt
> a*.txt
> ls a\?\*.txt
> a?*.txt
```

`> ls “a?*.txt”` or `ls 'a?*.txt'`

```
> a?*.txt
```

## History Expansion

- The history builtin
- The ! special character
  - `!!` Last command
  - `!n` Last command starting with letter n
  - …

- Speed and convenience for power users

## Aliases

- Shorthand for frequently used commands
  - Usually put them in your `~/.bashrc`
- Different from variables
- Syntax
  - Define alias: `alias ls=”ls --color”`
  - View alias: `alias ls`
  - Remove alias: `unalias ls`

## Readings

- Sections from the Linux Pocket Guide
    - Same sections as last lecture

- Class website lists additional resources