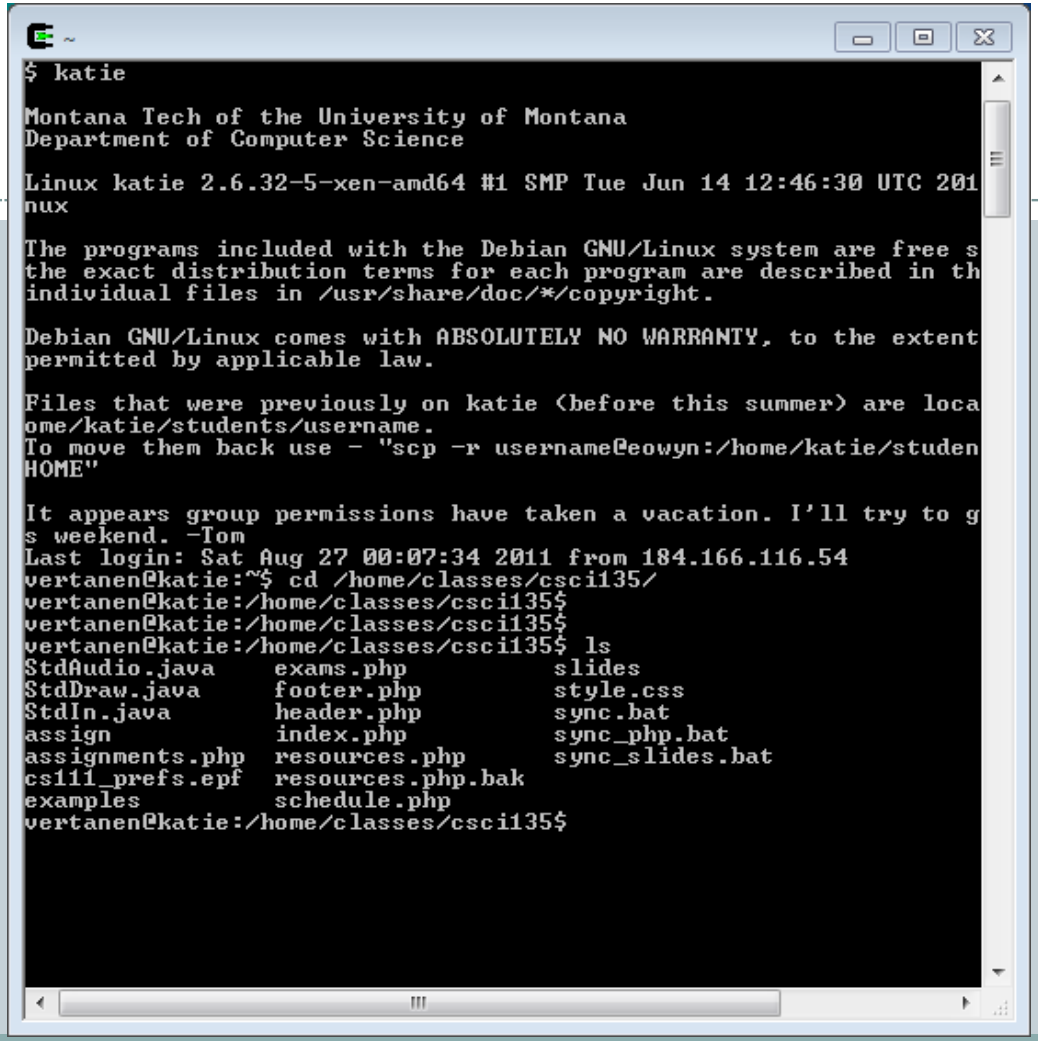
More Linux – Shell Scripts



Outline

- Shells and Shell Scripts
- Shell Variables and the Environment
- Simple Shell Scripting
- More Advanced Shell Scripting
- Start-up Shell Scripts

Shells and Shell Scripts

- Shell
  - Reads and executes commands for the user
  - Provides
    - Job control
    - Input / output redirection
    - Command language
  - Shell Script
    - Text file containing shell language commands
    - File can be executed as if commands were typed at the shell prompt

Shell Variables

- A variable is a piece of data that has a name
  - Just like in Java
- You can assign values to variables
  - `$ bob='hello world'`
- To access the variable's value, use a "$" at the beginning of the name
  - `$ echo $bob`

Shell Variables

- A variable is local to the current shell
  - If you open another shell, that variable won't be visible
- You can see what variables are available in a shell by:
  - `$ set`
- To make a variable available in all shells:
  - `$ export bob`
- To start another (Bourne again) shell
  - `$ bash`
- If you type ps, you will see you have two shells running now – and $bob is available in both

Some Handy Variables

- EDITOR
  - Tells Linux which editor to use by default
- PS1
  - Defines your command prompt
    - `export PS1="{\h} \w> "`
  - \h is hostname
  - \w current working directory
  - \d for the date
  - \t for the time
  - \u for user
- PATH
  - Tells Linux where to look for executable commands

Simple Shell Scripting

- Let's say the following text is in a file:

```
#!/bin/sh
# this is a comment
echo "The number of arguments is $#"
echo "The arguments are $*"
echo "The first is $1"
echo "My process number is $$"
echo "Enter a number from the keyboard: "
read number
echo "The number you entered was $number"
```

- # usually means comment, like // in Java
- But - #! is special
  - It must be the first two characters in the shell script
  - This one tells Linux to interpret this is a Bourne shell (sh)
- The next line really is a comment

Simple Shell Scripting

- Let's say the following text is in a file:

```
#!/bin/sh
# this is a comment
echo "The number of arguments is $#"
echo "The arguments are $*"
echo "The first is $1"
echo "My process number is $$"
echo "Enter a number from the keyboard: "
read number
echo "The number you entered was $number"
```

- Command line arguments come in as the variables $1, $2, $3, etc (like args[0], args[1], args[2] in Java)
- All arguments are $* (like args in Java)
- $# is the number of arguments (args.length in Java)
- $$ is the process number of the shell when it is running
- read allows you to read from standard input, 'number' is just a variable name

Creating the Script

- Use a text editor
  - In Linux can use vi, vim, nano, pico
  - Or (less elegantly) create the file in a text editor in Windows and copy it to the Linux machine
- Linux now considers this a text file
  - Need to make it executable
  - `chmod 644 simple.sh`
    - Gives me execute permission and all others get read permission
- To run the file, type:
  - `./simple.sh` OR
  - `simple.sh` (if . is in your PATH variable)
- Can now use the script with file redirection and pipes

More Advanced Shell Scripting

- if-then-else Statements

```
if [ test ]
then
     commands-if-test-is-true
else
     commands-if-test-is-false
fi
```

`-s` *file*
: true if *file* exists and is not empty

`-f` *file*
: true if *file* is an ordinary file

`-d` *file*
: true if *file* is a directory

`-r` *file*
: true if *file* is readable

`-w` *file*
: true if *file* is writable

`-x` *file*
: true if *file* is executable

`$X -eq $Y`
: true if `X` equals `Y`

`$X -ne $Y`
: true if `X` not equal to `Y`

`$X -lt $Y`
: true if `X` less than `$Y`

`$X -gt $Y`
: true `if` X greater than `$Y`

`$X -le $Y`
: true if `X` less than or equal to `Y`

`$X -ge $Y`
: true if `X` greater than or equal to `Y`

`"$A" = "$B"`
: true if string `A` equals string `B`

`"$A" != "$B"`
: true if string `A` not equal to string `B`

`$X ! -gt $Y`
: true if string `X` is not greater than `Y`

`$E -a $F`
: true if expressions `E` and `F` are both true

`$E -o $F`
: true `if` either expression `E` or expression `F` is true

More Advanced Shell Scripting

- for Loops

```
for variable in list
do
    statements  (referring to $variable)
done
```

```
#!/bin/sh
for f in *.txt
do
    echo sorting file $f
    cat $f | sort > $f.sorted
    echo sorted file has been output to $f.sorted
done
```

More Advanced Shell Scripting

- while Loops

```
while [ test ]
do
     statements     (to be executed while test is true)
done
```

```
#!/bin/sh
while [ ! -s input.txt ]
do
  echo waiting...
  sleep 5
done
echo input.txt is ready
```

```
#!/bin/sh
while true
do
  if [ -s input.txt ]
    echo input.txt is ready
    exit
  fi
  echo waiting...
  sleep 5
done
```

More Advanced Shell Scripting

- case Statements

```
case variable in
    pattern1)
        statement    (executed if variable matches pattern1)
        ;;
    pattern2)
        statement
        ;;
    etc.
esac
```

```
#!/bin/sh
for f in $*
do
  if [ -f $f -a ! -x $f ]
  then
    case $f in
    core)
      echo "$f: a core dump file"
      ;;
    *.c)
     echo "$f: a C program"
      ;;
    *.cpp|*.cc|*.cxx)
      echo "$f: a C++ program"
      ;;
    *.txt)
      echo "$f: a text file"
      ;;
    *.pl)
      echo "$f: a PERL script"
      ;;
    *.html|*.htm)
      echo "$f: a web document"
      ;;
    *)
      echo "$f: appears to be "`file -b $f`
      ;;
    esac
  fi
done
```

More Advanced Shell Scripting

- Capturing Command Output

```
#!\bin\sh
lines=`wc -l $1`
echo "the file $1 has $lines lines"
```

- Doing Arithmetic Operations

```
lines = `expr $lines + 1`
```

Start-Up Shell Scripts

- When you log in to a shell
  - First, a system-wide start-up script is executed
    - Usually /etc/profile
  - Then Linux looks in home directory for personal start-up script
    - .profile on katie
  - You can set any environment variables you would like in this script
  - If you modify your .profile and want it to take effect in the current shell:
    - `source .profile` OR
    - `. ./profile`

Summary

- Shells and Shell Scripts
- Shell Variables and the Environment
- Simple Shell Scripting
- More Advanced Shell Scripting
- Start-up Shell Scripts

Your Turn

- Recreate the simple shell script we saw on slide 7
  - Name this “simple”
  - Run this to make sure it works the same as the demonstration
- Modify this so that if the user enters a number greater than 10, the script prints out an error message “Number too large”
- Submit your modified shell script file to the Moodle dropbox for today
  - 1 EC point for turning something in, 2 EC points for turning in something correct