

## Shell I

### 1. Shell I

#### 1.1 Introduction

# CSCI 330 UNIX and Network Programming

---



Shell, Part 1



## 1.2 UNIX Command Interpreters

# UNIX Command Interpreters

common term: shell

- standard:
  - every UNIX system has a “Bourne shell compatible” shell
- history:
  - sh: original Bourne shell, written 1978 by Steve Bourne
  - ash: Almquist shell, BSD-licensed replacement of sh
- today:
  - bash: Bourne-again shell, GNU replacement of sh
  - dash: Debian Almquist shell, small scripting shell

## 1.3 bash shell basics

# bash shell basics

- Customization
  - variables, prompt and aliases
  - startup initialization
- Command line behavior
  - history
  - sequence & substitution
  - redirections and pipe

## 1.4 Variables

### Variables

---

- shell remembers values in variables
- variable has name and type: string, number, array
- to set string variable:

Syntax:

```
% varname=value
```

- to display variable's value

```
% echo $varname
```

## 1.5 Variables

### Variables

---

Examples:

```
% speed=fast
% echo Today we go $speed
Today we go fast
% speed="very fast"
% echo Now we go $speed
Now we go very fast
```

## 1.6 Variable Scope

### Variable Scope

- variable holds value for duration of shell invocation
- variable can be exported into environment:  
inherited by commands, shell scripts and subshells

Term: *environment variable*

Example: `% export fast`

## 1.7 some predefined variables

### some predefined variables

Name	Meaning
HOME	full pathname of your home directory
PATH	list of directories to search for commands
USER	Your user name, also UID for user id
SHELL	full pathname of your <u>login shell</u>
PWD	Current work directory
HOSTNAME	current hostname of the system
HISTSIZE	Number of commands to remember
PS1	primary prompt (also PS2, ...)
?	Return status of most recently executed command
\$	Process id of current process

## 1.8 Example: PATH variable

### Example: PATH variable

- PATH lists a set of directories
- shell finds commands in these directories

#### Example:

```
% echo $PATH
/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
% PATH=$PATH:~/bin
% echo $PATH
/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/home/
student/bin
```

## 1.9 bash shell prompt

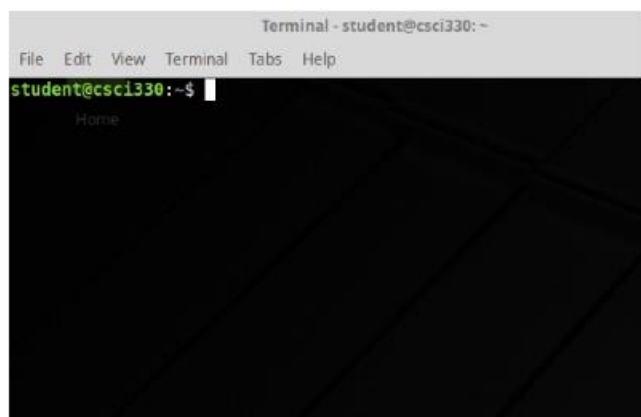
### bash shell prompt

- can be set via “PS1” shell variable

#### Example:

```
% PS1="$USER > "
student >
```

- Secondary prompts:  
PS2, PS3, PS4



## 1.10 bash shell prompt

### bash shell prompt

- special “PS1” shell variable settings:

\w    current work directory

\h    hostname

\u    username

\d    date

\t    time

\a    ring the “bell”

Example:

```
% PS1="\u@\h \w \$ "
```

```
student@csci330 ~ $
```

## 1.11 shell aliases

### shell aliases

- Allows you to assign a different name to a command
  - use alias like any other command

- to check current aliases:

```
% alias
```

- to set alias:

```
% alias ll="ls -al"
```

- to remove alias:

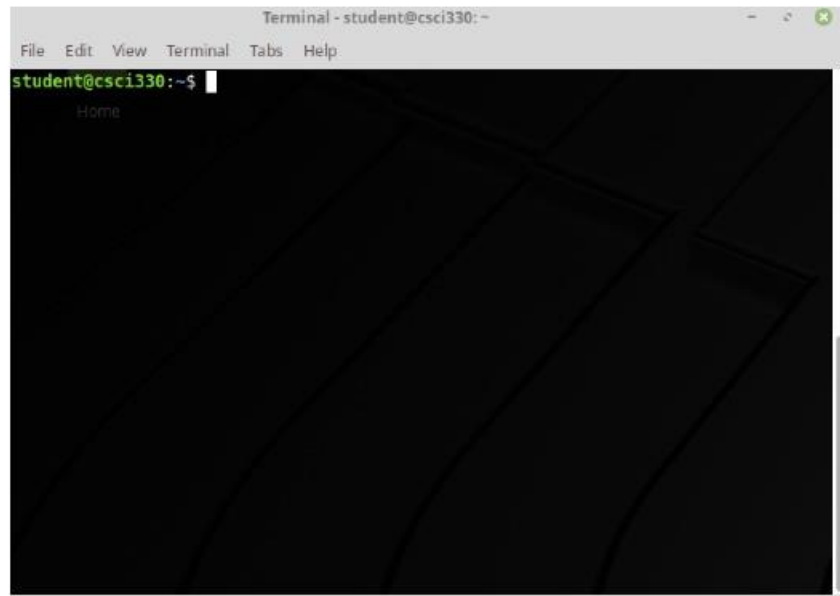
```
% unalias ll
```

### 1.12 How to set and keep variables ?

## How to set and keep variables ?

- variables set on the command line end when shell ends
- set variables via a text file ?
  - enter alias definitions into text file
    - if text file is run as shell script, variables are local to that invocation
  - execute text file via "source" or "." command
    - reads and executes content of file in current shell
- solution: set variables via default startup files

### 1.13 Setting aliases





## 1.14 Customization

### Customization

---

- via command line options
  - rarely done
- instead: startup initialization file

`~/.profile`      if login session shell

`~/.bashrc`      if invoked from command line

- Also: `/etc/profile` and `/etc/bash.bashrc`

## 1.15 Command line behavior

### Command line behavior

---

- history
- sequence
- substitution
- I/O redirection and pipe



## 1.16 Shell History

### Shell History

- record of previously entered commands
  - can be: re-called, edited, re-executed
- commands are saved: size of history is set via shell variables
  - per session `HISTSIZE=500`
  - per user `HISTFILESIZE=100`
- to view the history buffer:

Syntax: `history [-c] [count]`

## 1.17 Command line editing

### Command line editing

- ↑ UP ARROW  
move back one command in history list
- ↓ DOWN ARROW  
move forward one command
- ← LEFT and → RIGHT ARROW  
move into command
- BACKSPACE and DELETE  
remove information
- TAB  
complete current command or file name

### 1.18 Command Sequence

## Command Sequence

- allows series of commands all at once
- commands are separated by a semicolon ;

Example:

```
% date; pwd; ls
```

### 1.19 Command Substitution

## Command Substitution

- command surrounded by back quotes ` ` is run and replaced by its standard output
- newlines in the output are replaced by spaces

Examples:

```
% ls -l `which passwd`
```

```
% var=`whoami`; echo $var
```

## 1.20 Command Substitution

### Command Substitution

---

- second form of command substitution:  
`$(command)`

#### Examples:

```
% echo User $(whoami) is on $(hostname)
User student is on niu
% echo Today is $(date)
Today is Wed Feb 12 10:32:23 CST 2014
```

## 1.21 Output Redirection (>)

### Output Redirection (>)

---

Syntax: `command > file`

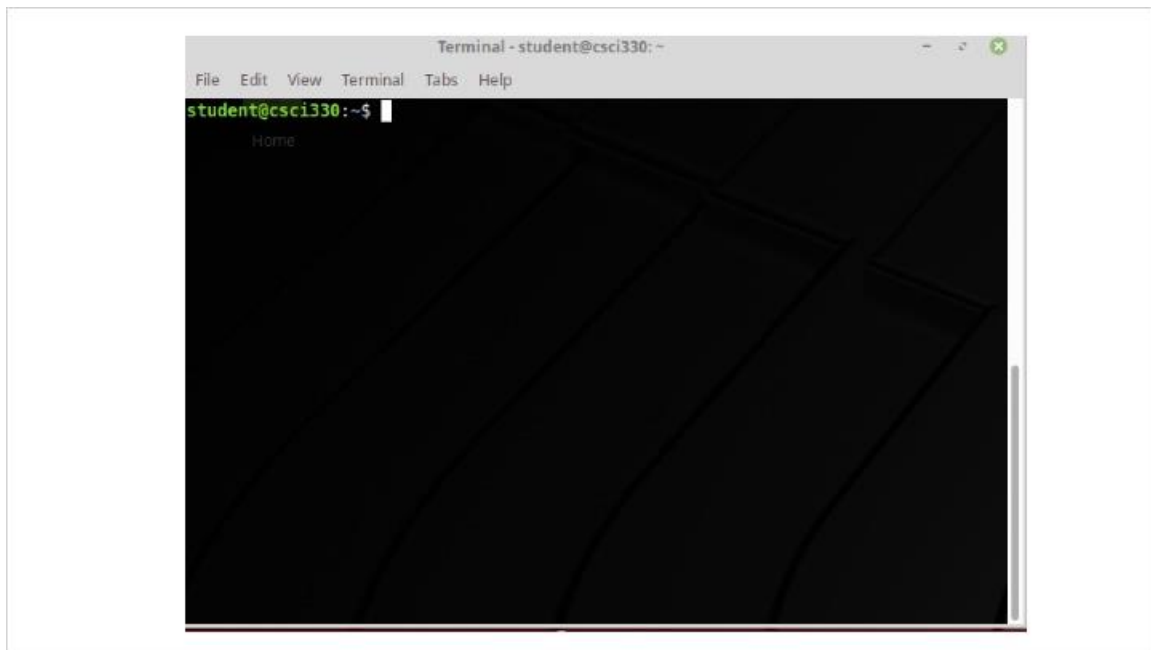
sends command output to file, instead of terminal

#### Examples:

```
% ls > listing
% cat listing > filecopy
```

Note: if “file” exists, it is overwritten

## 1.22 Redirect example



## 1.23 Input Redirection (<)

### Input Redirection (<)

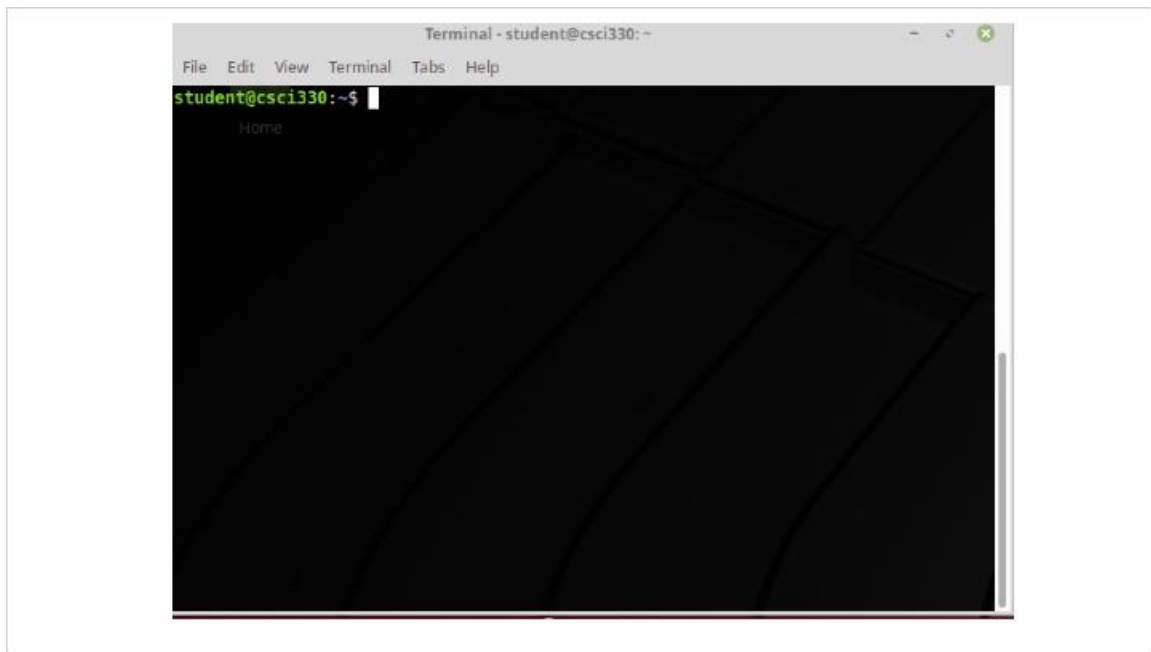
Syntax: `command < file`

command reads (takes input) from file,  
instead of keyboard

Example:

```
% tr a-z A-Z < listing
```

## 1.24 Rediirect example

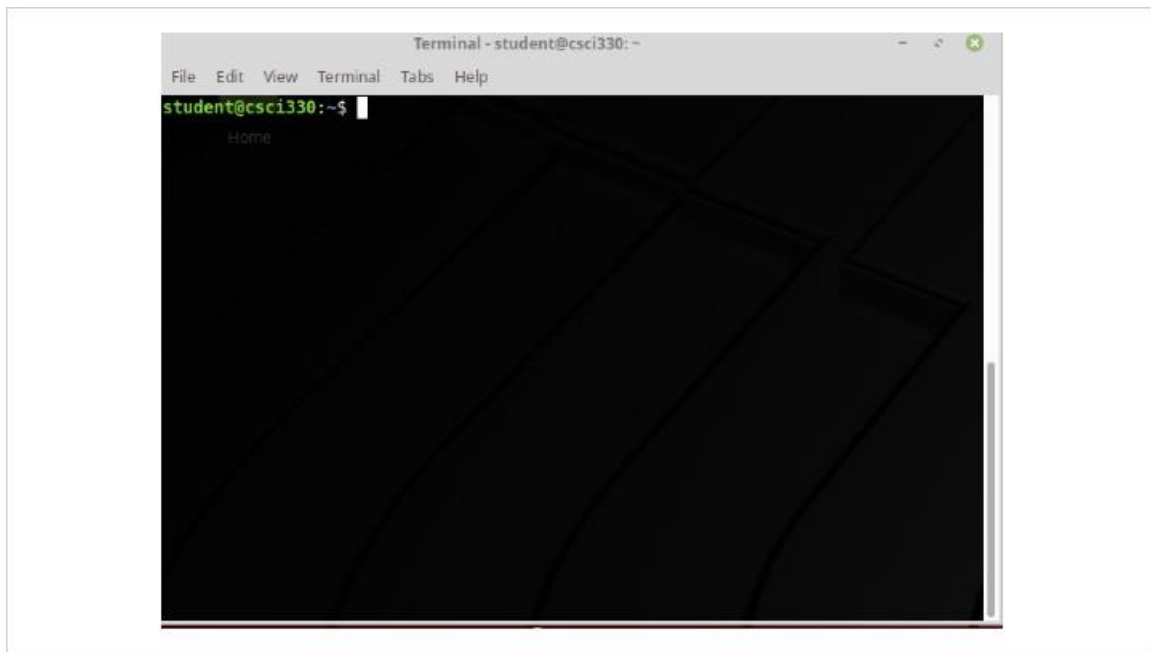


## 1.25 Examples: Output / Input

### Examples: Output / Input

- Redirecting input and output:  
`% tr A-Z a-z < r.in > r.out`
- Output of command becomes input to next:  
`% ls > /tmp/out.txt; wc < /tmp/out.txt`
- Eliminate the middleman: pipe  
`% ls | wc`

## 1.26 Pipe example



## 1.27 Appending Output

### Appending Output

- Syntax: **command >> file**  
adds output of command at the end of file
  - If file does not exist, shell creates it
- Examples:
  - % `date > usage-status`
  - % `ls -l >> usage-status`
  - % `du -s >> usage-status`

} Build the file 'usage-status' from the output of 'date', 'ls', and 'du'

## 1.28 Here Document

### Here Document

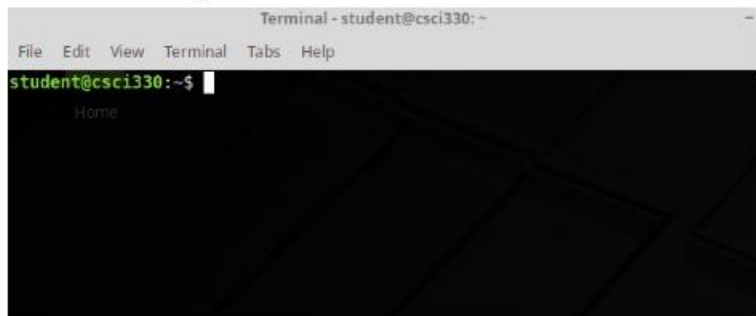
- read input for current source, uses "<<" symbol

Syntax: **command << LABEL**

- reads following lines until line starting with "LABEL"

Example:

```
% wc -l << DONE
> line one
> line two
> DONE
2
```



## 1.29 File Descriptor

### File Descriptor

- positive integer for every open file
  - process tracks its open files with this number
- 
- 0 – standard input
  - 1 – standard output
  - 2 – standard error output
- 
- bash can use file descriptor to refer to a file



### 1.30 Redirection syntax

## Redirection syntax

- Output:  
`> or 1> filename`  
`2> filename`
- Input:  
`< or 0<`
- Combining outputs:  
`2>&1 or &> or >&`

#### Example:

`% cat mouse > /tmp/out.txt 2>&1`  
or `% cat mouse &> /tmp/out.txt`

### 1.31 Summary: Redirections and Pipe

## Summary: Redirections and Pipe

Command Syntax	Meaning
<code>command &lt; file</code>	redirect input from <i>file</i> to <i>command</i>
<code>command &gt; file</code>	redirect output from <i>command</i> to <i>file</i>
<code>command &gt;&gt; file</code>	redirect output of <i>command</i> and appends it to <i>file</i>
<code>command &gt; file 2&gt;&amp;1</code> <code>command &amp;&gt; file</code>	add error output to standard output, redirect both into <i>file</i>
<code>command1  </code> <code>command2</code>	take/pipe output of <i>command1</i> as input to <i>command2</i>
<code>command &lt;&lt; LABEL</code>	take input from current source until <i>LABEL</i> line

## 1.32 Summary

### Summary

---

- features of the UNIX shell:
  - customization
    - variables, prompt, alias, startup files
  - command line behavior
    - history
    - sequence, substitution
    - redirections and pipe