# Lecture 4

*Useful commands, I/O redirection*

COP 3344 Introduction to UNIX
Fall 2007

Acknowledgment: These slides are modified versions of Prof. Sudhir Aggarwal's slides



## Standard input, output, and error

- standard input (*0: stdin*)
  - The default place where a process reads its input (usually the terminal keyboard)
- standard output (*1: stdout*)
  - The default place where a process writes its output (typically the terminal display)
- standard error (*2: stderr*)
  - the default place where a process can send its error messages (typically the terminal display)



## Redirecting standard I/O

- Standard input and output can be redirected
  - Lets you combine programs and Unix tools
- Standard input is redirected from a file using <
  `wc < input12`
  - Any use of *stdin* will instead use `input12` in this example

input12

```
f af asfsdf afdsf
sdf gvddfg ssdffg
sdgf asff assdf
```

```
$ wc < input12
   3  10   54
```

– Output
  – Lines Words Characters



## Redirecting stdout

- Standard output is redirected to a file using >

```
cal 2007 > calendar.2007
wc -l < input12 > Lines.input12
```

  – the *stdout* of *wc -l* is directed to file `LinesIn.input12`

```
$ wc -l < input12 > Lines.input12
$ cat Lines.input12
   3
```



## Redirecting stderr

- Standard error can be redirected with 2> (sh)
- Standard error and standard out can be simultaneously redirected with >& (csh)

```
$ ls
   Lines.input12 input12
$ ls Line.*
   ls: Line.*: No such
      file or directory
$ ls Line.* > LS.output
   ls: Line.*: No such
      file or directory
```

```
$ cat LS.output
$
$ ls Line.* 2> LS.output
$ cat LS.output
   ls: Line.*: No such
      file or
      directory
```



## Appending to a file

- The >> operator *appends* to a file rather than redirecting the output to a file

```
$ cal 2007 > calendar.txt
$cal 2008 >> calendar.txt
$cal 2009 >> calendar.txt
$ head calendar.txt
                             2007

      January               February
S  M Tu  W Th  F  S   S  M Tu  W Th  F  S
   1  2  3  4  5  6               1  2  3
7  8  9 10 11 12 13   4  5  6  7  8  9 10 ...
14 15 16 17 18 19 20  11 12 13 14 15 16 17
21 22 23 24 25 26 27  18 19 20 21 22 23 24
28 29 30 31           25 26 27 28
```

```
$ tail calendar.txt
                                30 31
      October                 November
S  M Tu  W Th  F  S   S  M Tu  W Th  F  S
                1  2   1  2  3  4  5  6  7
 4  5  6  7  8  9 10   8  9 10 11 12 13 14 ...
11 12 13 14 15 16 17  15 16 17 18 19 20 21
18 19 20 21 22 23 24  22 23 24 25 26 27 28
25 26 27 28 29 30 31  29 30
```

## Pipes

- Pipes allow *stdout* of one program to be stdin of another
  - The output of the command to the left of '|' becomes the input of the command on its right
- Examples

```
ls | sort
cat < input* | wc -l
```

```
$ ls -l | cut -c 40-80
    Sep 23 13:54 Lines.input12
    Sep 23 14:24 calendar.txt
    Sep 23 13:40 input12
```

```
$ ls -l
    -rw-r--r--   1 asriniva  asriniva     9 Sep 23 13:54 Lines.input12
    -rw-r--r--   1 asriniva  asriniva  5841 Sep 23 14:24 calendar.txt
    -rw-r--r--   1 asriniva  asriniva    54 Sep 23 13:40 input12
```



## Separating commands

- Multiple instructions on one line
  - separate instructions by ';'

```
ls -l; cal; date
```

- Continue a command on the next line using '\'

```
(cal 2007; cal 2008; cal 2009) \
> calendar.txt
```



## diff

- *diff* compares two text files
  - It prints the lines that differ the
  - `diff [options] <original file> <newfile>`
    - `[-i]` Ignores case
    - `[-w]` Ignores all spaces and tabs

testprog1.c

```
#include <stdio.h>

int Main()
{
  printf("Hello
    World!\n");

  return 0;
}
```

testprog2.c

```
#include <stdio.h>
#include <assert.h>

int main()
{
      printf("Hello
    World!\n");
}
```

```
$ diff -w testprog1.c testprog2.c
1a2
> #include <assert.h>
3,4c4
<
< int Main()
---
> int main()
7,8d6
<
<   return 0;
```



## cmp and gzip

- *cmp*
  - Tells where two files differ

```
$ cmp myfile1.o myfile2.o
    myfile1.o myfile2.o differ: char 20, line 2
```

- *gzip / gunzip*
  - Compress/uncompress files
  - Examples to compress and restore a file called *bigfile*
    - `gzip bigfile` (The compressed file has extension *.gz*)
    - `gzip -d bigfile.gz` (this restores a *.gz* file)
    - `gunzip bigfle.gz` (same as the line above)



## tar

- Tar is a utility for creating and extracting archives
  - `tar [options] filenames`
- Commonly used Options
  - c insert files into a tar file
  - f use the name of the tar file that is specified
  - v verbose -- output file names
  - x extract the files from a tar file
  - *t* list contents of an archive
  - *z* will *gzip / gunzip* if necessary



## Creating an Archive with Tar

- Note: *tar*-ing a directory will insert all files in subdirectories too
- Example

```
tar cvf HW2.tar HW2
```

  - If HW2 is a directory this will insert the directory and all files and subdirectories (recursively) into the archive *HW2.tar*

```
$ ls *.[ch]
    testprog1.c testprog2.c
$ ls HW2
    testprog1.c testprog2.c
```

```
$ tar cvf HW2.tar HW2 *.[ch]
    HW2/
    HW2/testprog1.c
    HW2/testprog2.c
    testprog1.c
    testprog2.c
```

# Extracting from tar files

- The extracted files have the same permissions, name, and directory structure as the original files
  - If they are opened by another user (archive sent by email) the user id becomes that of the user opening the tar archive
- Example

  `tar xvf HW2.tar`
  - Extract all files from *proj.tar* into the current directory

```
$ gzip HW2.tar
...
$ tar xvzf HW2.tar.gz
      HW2/
      HW2/testprog1.c
      HW2/testprog2.c
      testprog1.c
      testprog2.c
```

13