Filters

ComS 252 — Iowa State University

Andrew Miner

Remember pipelines?

In UNIX, it is common to do things like

```
prompt$ cmdA args | cmdB | cmdC | cmdD | cmdE
```

What goes in the middle?

- Utility that
  - Reads from `stdin`
  - Writes to `stdout`
  - Does something interesting in between
- Think of these as “filters”
- These utilities give you lots of power
  - Remember: power of UNIX comes from simple utilities + ability to combine them (with pipes)
- We will cover simple ones for now

We will also discuss utilities that go in front or at the end

Pagers

- Suppose output from some command is too long; e.g.,

```
prompt$ ps aux
```

- How to see the “top part” of the output?

Pagers

- Suppose output from some command is too long; e.g.,

```
prompt$ ps aux
```

- How to see the “top part” of the output?
- Easy — pipe output into a pager

more: the classic pager

- Usage: `more [file]`
  - If no file is given, reads from standard input
- Displays file until terminal is full
- When you press a key, get next screen of output
- This continues until either
  - You quit (usually, q)
  - All output has been displayed

```
prompt$ █
```

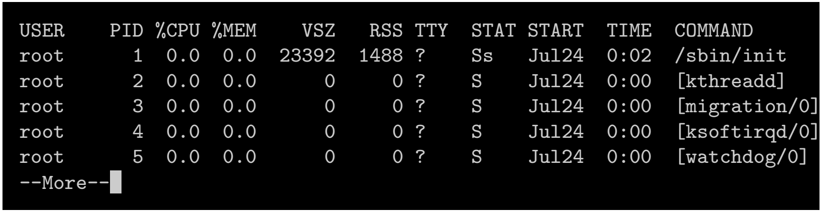
more: the classic pager

- Usage: `more [file]`
  - If no file is given, reads from standard input
- Displays file until terminal is full
- When you press a key, get next screen of output
- This continues until either
  - You quit (usually, q)
  - All output has been displayed

```
prompt$ ps aux | more
```

more: the classic pager

- Usage: `more [file]`
  - If no file is given, reads from standard input
- Displays file until terminal is full
- When you press a key, get next screen of output
- This continues until either
  - You quit (usually, q)
  - All output has been displayed

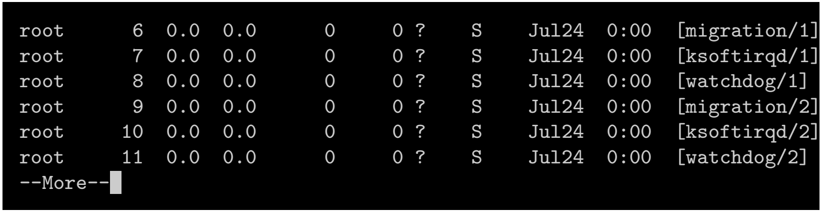


```
USER      PID %CPU %MEM    VSZ   RSS TTY    STAT START  TIME  COMMAND
root        1  0.0  0.0  23392  1488 ?      Ss   Jul24  0:02  /sbin/init
root        2  0.0  0.0      0     0 ?      S    Jul24  0:00  [kthreadd]
root        3  0.0  0.0      0     0 ?      S    Jul24  0:00  [migration/0]
root        4  0.0  0.0      0     0 ?      S    Jul24  0:00  [ksoftirqd/0]
root        5  0.0  0.0      0     0 ?      S    Jul24  0:00  [watchdog/0]
--More--
```

Press space for next page

more: the classic pager

- Usage: `more [file]`
  - If no file is given, reads from standard input
- Displays file until terminal is full
- When you press a key, get next screen of output
- This continues until either
  - You quit (usually, q)
  - All output has been displayed



```
root       6  0.0  0.0       0     0 ?     S     Jul24  0:00  [migration/1]
root       7  0.0  0.0       0     0 ?     S     Jul24  0:00  [ksoftirqd/1]
root       8  0.0  0.0       0     0 ?     S     Jul24  0:00  [watchdog/1]
root       9  0.0  0.0       0     0 ?     S     Jul24  0:00  [migration/2]
root      10  0.0  0.0       0     0 ?     S     Jul24  0:00  [ksoftirqd/2]
root      11  0.0  0.0       0     0 ?     S     Jul24  0:00  [watchdog/2]
--More--
```

Press q to quit

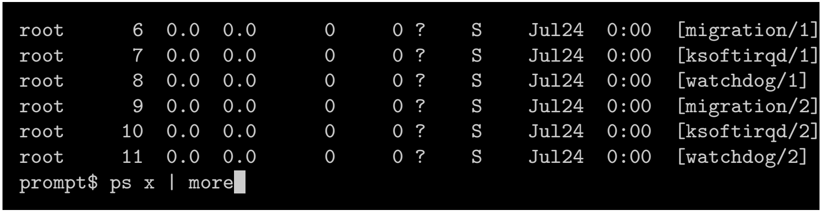
more: the classic pager

- Usage: `more [file]`
  - If no file is given, reads from standard input
- Displays file until terminal is full
- When you press a key, get next screen of output
- This continues until either
  - You quit (usually, q)
  - All output has been displayed

```
root        6  0.0  0.0        0       0 ?     S     Jul24  0:00  [migration/1]
root        7  0.0  0.0        0       0 ?     S     Jul24  0:00  [ksoftirqd/1]
root        8  0.0  0.0        0       0 ?     S     Jul24  0:00  [watchdog/1]
root        9  0.0  0.0        0       0 ?     S     Jul24  0:00  [migration/2]
root       10  0.0  0.0        0       0 ?     S     Jul24  0:00  [ksoftirqd/2]
root       11  0.0  0.0        0       0 ?     S     Jul24  0:00  [watchdog/2]
prompt$
```

more: the classic pager

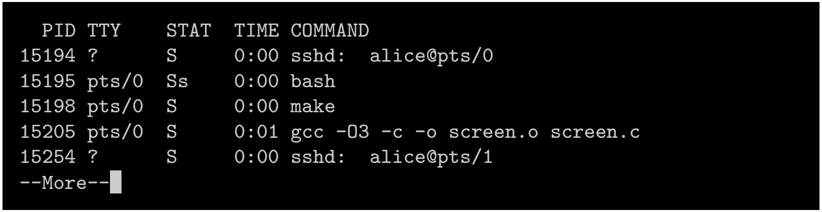
- Usage: `more [file]`
  - If no file is given, reads from standard input
- Displays file until terminal is full
- When you press a key, get next screen of output
- This continues until either
  - You quit (usually, q)
  - All output has been displayed



```
root        6  0.0  0.0        0       0 ?     S     Jul24   0:00  [migration/1]
root        7  0.0  0.0        0       0 ?     S     Jul24   0:00  [ksoftirqd/1]
root        8  0.0  0.0        0       0 ?     S     Jul24   0:00  [watchdog/1]
root        9  0.0  0.0        0       0 ?     S     Jul24   0:00  [migration/2]
root       10  0.0  0.0        0       0 ?     S     Jul24   0:00  [ksoftirqd/2]
root       11  0.0  0.0        0       0 ?     S     Jul24   0:00  [watchdog/2]
prompt$ ps x | more
```

more: the classic pager

- Usage: `more [file]`
  - If no file is given, reads from standard input
- Displays file until terminal is full
- When you press a key, get next screen of output
- This continues until either
  - You quit (usually, q)
  - All output has been displayed

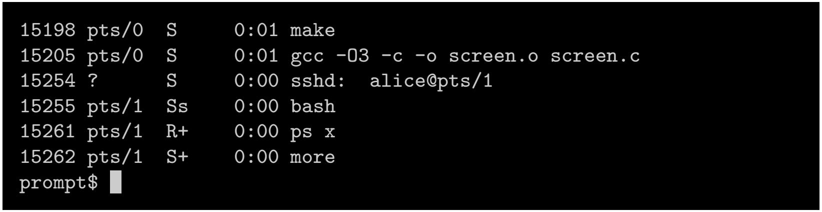


```
  PID TTY     STAT   TIME COMMAND
15194 ?       S      0:00 sshd:  alice@pts/0
15195 pts/0   Ss     0:00 bash
15198 pts/0   S      0:00 make
15205 pts/0   S      0:01 gcc -O3 -c -o screen.o screen.c
15254 ?       S      0:00 sshd:  alice@pts/1
--More--
```

Press space for next page

more: the classic pager

- Usage: `more [file]`
  - If no file is given, reads from standard input
- Displays file until terminal is full
- When you press a key, get next screen of output
- This continues until either
  - You quit (usually, q)
  - All output has been displayed



```
15198 pts/0   S      0:01 make
15205 pts/0   S      0:01 gcc -O3 -c -o screen.o screen.c
15254 ?       S      0:00 sshd:  alice@pts/1
15255 pts/1   Ss     0:00 bash
15261 pts/1   R+     0:00 ps x
15262 pts/1   S+     0:00 more
prompt$ 
```

`less`: a nicer pager

- Usage: `less [file]`
- Works like `more`, but has more features
  - Can scroll up or down
  - Can search for text
    - Use `/text` to search forward
    - Use `?text` to search backward
  - Many of the keystrokes are the same as `vi`
    - This happens often in UNIX
- Fun fact: the `man` pages are piped through `less`
  - Super fun fact: you can set `man` to use a different pager
- But why the name?

less: a nicer pager

- Usage: `less [file]`
- Works like `more`, but has more features
  - Can scroll up or down
  - Can search for text
    - Use `/text` to search forward
    - Use `?text` to search backward
  - Many of the keystrokes are the same as `vi`
    - This happens often in UNIX
- Fun fact: the `man` pages are piped through `less`
  - Super fun fact: you can set `man` to use a different pager
- But why the name?

  "`less` is `more`, more or less"

Filters that select certain lines

head: select the beginning of a file

- Usage: `head [-n count] [file]`
- Write the first `count` lines of `file` to standard output
- If no `count` is specified: 10 lines
- If no `file` is specified: use standard input
- Can also print the first few *bytes* of the file
  - `head -c bytes`

```
prompt$ █
```

Filters that select certain lines

head: select the beginning of a file

- Usage: `head [-n count] [file]`
- Write the first `count` lines of `file` to standard output
- If no `count` is specified: 10 lines
- If no `file` is specified: use standard input
- Can also print the first few *bytes* of the file
  - `head -c bytes`

```
prompt$ ps aux | head -n 4
```

Filters that select certain lines

head: select the beginning of a file

- Usage: `head [-n count] [file]`
- Write the first `count` lines of `file` to standard output
- If no `count` is specified: 10 lines
- If no `file` is specified: use standard input
- Can also print the first few *bytes* of the file
  - `head -c bytes`

```
prompt$ ps aux | head -n 4
USER       PID %CPU %MEM    VSZ   RSS TTY   STAT START  TIME  COMMAND
root         1  0.0  0.0  23392  1488 ?     Ss   Jul24  0:02  /sbin/init
root         2  0.0  0.0      0     0 ?     S    Jul24  0:00  [kthreadd]
root         3  0.0  0.0      0     0 ?     S    Jul24  0:00  [migration/0]
prompt$ 
```

`tail`: select the end of a file

- Usage: `tail [-n count] [file]`
- Writes the end of `file` to standard output
- If no `file` is specified: use standard input
- Two ways to specify "how many lines":

  `-n +count` : start writing *count* lines from the top
  - Lines $1, \ldots, count - 1$ are not written
  - Lines $count, \ldots$ are written

  `-n -count` : start writing *count* lines from the bottom
  - The last *count* lines of the file are written
- Note that `-n count` behaves like `-n -count`
- The default is `-n 10`

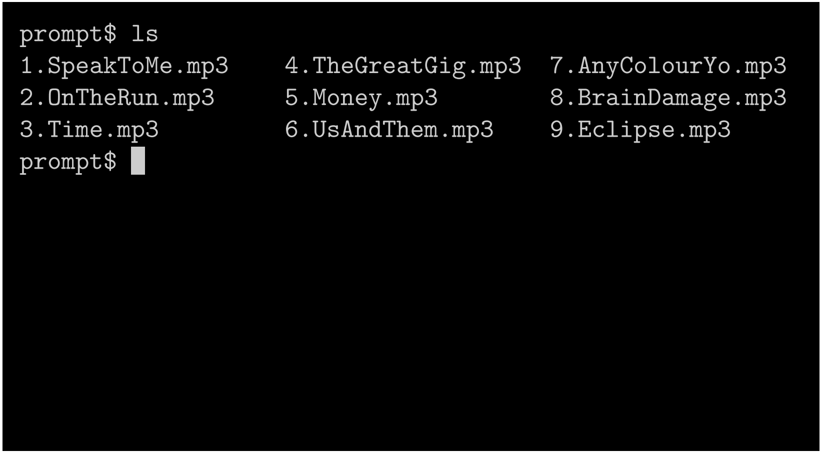
tail examples

```
prompt$
```

tail examples

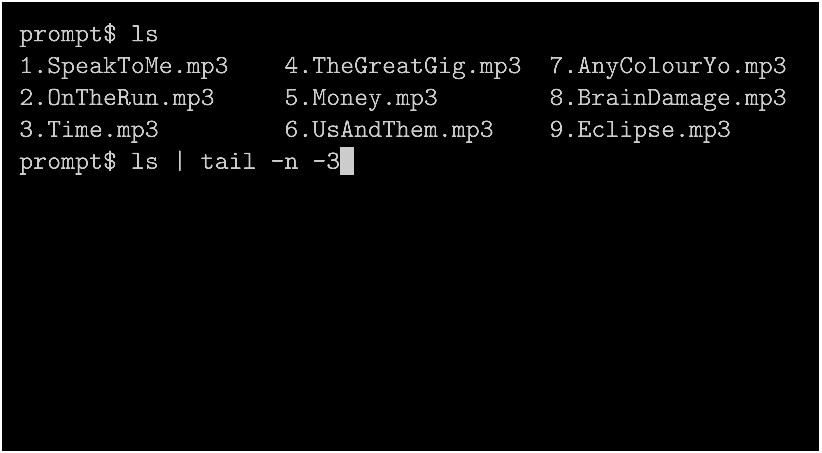
```
prompt$ ls
```

tail examples



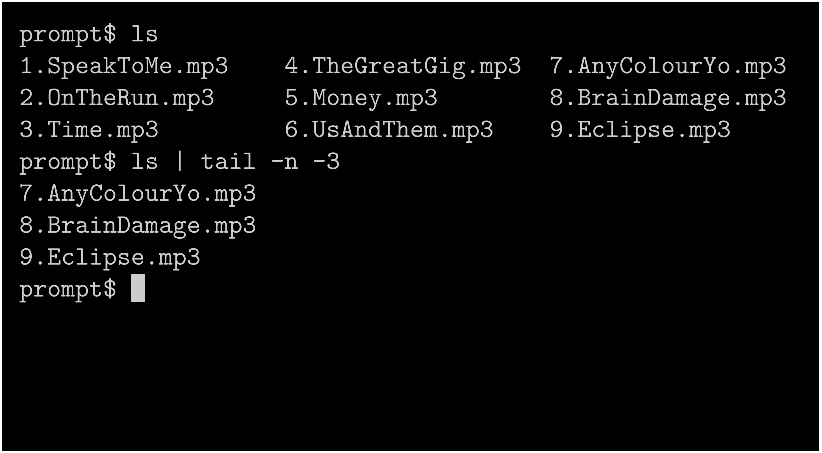
```
prompt$ ls
1.SpeakToMe.mp3     4.TheGreatGig.mp3  7.AnyColourYo.mp3
2.OnTheRun.mp3      5.Money.mp3        8.BrainDamage.mp3
3.Time.mp3          6.UsAndThem.mp3    9.Eclipse.mp3
prompt$ 
```

tail examples



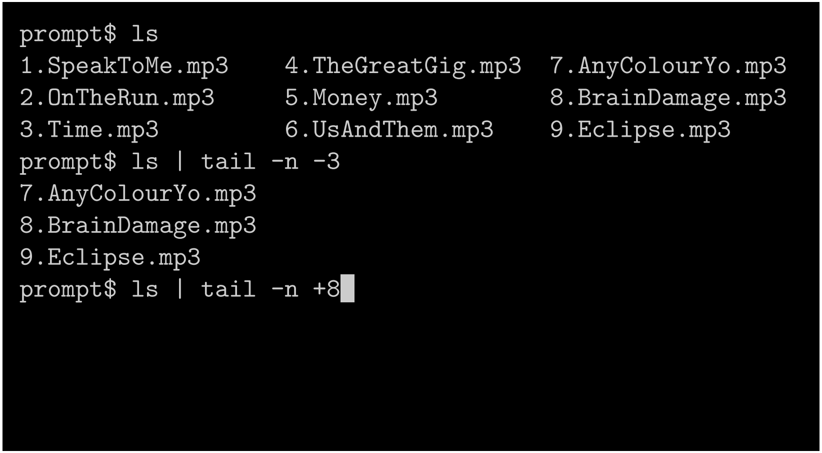
```
prompt$ ls
1.SpeakToMe.mp3     4.TheGreatGig.mp3  7.AnyColourYo.mp3
2.OnTheRun.mp3      5.Money.mp3        8.BrainDamage.mp3
3.Time.mp3          6.UsAndThem.mp3    9.Eclipse.mp3
prompt$ ls | tail -n -3
```

tail examples



```
prompt$ ls
1.SpeakToMe.mp3    4.TheGreatGig.mp3  7.AnyColourYo.mp3
2.OnTheRun.mp3     5.Money.mp3        8.BrainDamage.mp3
3.Time.mp3         6.UsAndThem.mp3    9.Eclipse.mp3
prompt$ ls | tail -n -3
7.AnyColourYo.mp3
8.BrainDamage.mp3
9.Eclipse.mp3
prompt$ 
```

tail examples



```
prompt$ ls
1.SpeakToMe.mp3     4.TheGreatGig.mp3  7.AnyColourYo.mp3
2.OnTheRun.mp3      5.Money.mp3        8.BrainDamage.mp3
3.Time.mp3          6.UsAndThem.mp3    9.Eclipse.mp3
prompt$ ls | tail -n -3
7.AnyColourYo.mp3
8.BrainDamage.mp3
9.Eclipse.mp3
prompt$ ls | tail -n +8
```

tail examples

```
prompt$ ls
1.SpeakToMe.mp3    4.TheGreatGig.mp3  7.AnyColourYo.mp3
2.OnTheRun.mp3     5.Money.mp3        8.BrainDamage.mp3
3.Time.mp3         6.UsAndThem.mp3    9.Eclipse.mp3
prompt$ ls | tail -n -3
7.AnyColourYo.mp3
8.BrainDamage.mp3
9.Eclipse.mp3
prompt$ ls | tail -n +8
8.BrainDamage.mp3
9.Eclipse.mp3
prompt$ █
```

tail examples

```
prompt$ ls
1.SpeakToMe.mp3    4.TheGreatGig.mp3  7.AnyColourYo.mp3
2.OnTheRun.mp3     5.Money.mp3        8.BrainDamage.mp3
3.Time.mp3         6.UsAndThem.mp3    9.Eclipse.mp3
prompt$ ls | tail -n -3
7.AnyColourYo.mp3
8.BrainDamage.mp3
9.Eclipse.mp3
prompt$ ls | tail -n +8
8.BrainDamage.mp3
9.Eclipse.mp3
prompt$ ls | tail -n +25
```

tail examples

```
prompt$ ls
1.SpeakToMe.mp3    4.TheGreatGig.mp3  7.AnyColourYo.mp3
2.OnTheRun.mp3     5.Money.mp3        8.BrainDamage.mp3
3.Time.mp3         6.UsAndThem.mp3    9.Eclipse.mp3
prompt$ ls | tail -n -3
7.AnyColourYo.mp3
8.BrainDamage.mp3
9.Eclipse.mp3
prompt$ ls | tail -n +8
8.BrainDamage.mp3
9.Eclipse.mp3
prompt$ ls | tail -n +25
prompt$ 
```

Pro tip: `tail -f`

- "Follows" the end of the file
- Does not exit when end of file is reached
- Instead, waits for and displays more lines as they are added
- Useful for watching a file (say, a log file)
- Stops with Ctrl–C
- Does not work with a pipe

Terminal

```
prompt$ 
```

Terminal

```
prompt$ tail -n 5 -f grail.txt
```

Terminal

```
prompt$ tail -n 5 -f grail.txt
Arthur: Old woman!
Dennis: Man!
Arthur: Man, Sorry.  What knight lives
        in that castle over there?
Dennis: I'm 37.
█
```

Terminal

```
prompt$ tail -n 5 -f grail.txt
Arthur: Old woman!
Dennis: Man!
Arthur: Man, Sorry.  What knight lives
        in that castle over there?
Dennis: I'm 37.
▯
```

Terminal

```
prompt$ █
```

Terminal

```
prompt$ tail -n 5 -f grail.txt
Arthur: Old woman!
Dennis: Man!
Arthur: Man, Sorry.  What knight lives
        in that castle over there?
Dennis: I'm 37.
```

Terminal

```
prompt$ echo "Arthur: What?" >> grail.txt
```

Terminal

```
prompt$ tail -n 5 -f grail.txt
Arthur: Old woman!
Dennis: Man!
Arthur: Man, Sorry.  What knight lives
        in that castle over there?
Dennis: I'm 37.
Arthur: What?
```

Terminal

```
prompt$ echo "Arthur: What?" >> grail.txt
prompt$
```

Terminal

```
prompt$ tail -n 5 -f grail.txt
Arthur: Old woman!
Dennis: Man!
Arthur: Man, Sorry.  What knight lives
        in that castle over there?
Dennis: I'm 37.
Arthur: What?
```

Terminal

```
prompt$ echo "Arthur: What?" >> grail.txt
prompt$ echo "Dennis: I'm 37, I'm not old." >> grail.txt
```

Terminal

```
prompt$ tail -n 5 -f grail.txt
Arthur: Old woman!
Dennis: Man!
Arthur: Man, Sorry.  What knight lives
        in that castle over there?
Dennis: I'm 37.
Arthur: What?
Dennis: I'm 37, I'm not old.
█
```

Terminal

```
prompt$ echo "Arthur: What?" >> grail.txt
prompt$ echo "Dennis: I'm 37, I'm not old." >> grail.txt
prompt$ █
```

Terminal

```
prompt$ tail -n 5 -f grail.txt
Arthur: Old woman!
Dennis: Man!
Arthur: Man, Sorry.  What knight lives
        in that castle over there?
Dennis: I'm 37.
Arthur: What?
Dennis: I'm 37, I'm not old.
```

Terminal

```
prompt$ echo "Arthur: What?" >> grail.txt
prompt$ echo "Dennis: I'm 37, I'm not old." >> grail.txt
prompt$ exit
```

Terminal

```
prompt$ tail -n 5 -f grail.txt
Arthur: Old woman!
Dennis: Man!
Arthur: Man, Sorry.  What knight lives
        in that castle over there?
Dennis: I'm 37.
Arthur: What?
Dennis: I'm 37, I'm not old.
█
```

Terminal

```
prompt$ tail -n 5 -f grail.txt
Arthur: Old woman!
Dennis: Man!
Arthur: Man, Sorry.  What knight lives
        in that castle over there?
Dennis: I'm 37.
Arthur: What?
Dennis: I'm 37, I'm not old.
^C
prompt$ 
```

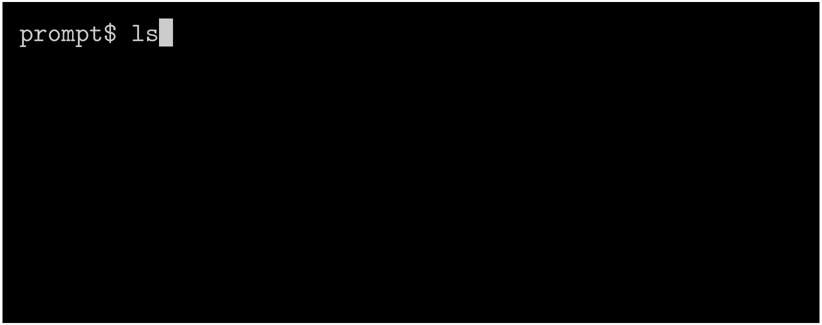
grep: select lines that match a pattern

- Usage: `grep pattern [file] [file] ...`
- No files specified: reads from standard input
  - Do you see a pattern yet?
- Simple pattern: plain text
- Fancier patterns: later in the semester
- Writes lines that contain the pattern

```
prompt$ 
```

grep: select lines that match a pattern

- Usage: `grep pattern [file] [file] ...`
- No files specified: reads from standard input
  - Do you see a pattern yet?
- Simple pattern: plain text
- Fancier patterns: later in the semester
- Writes lines that contain the pattern



```
prompt$ ls
```

grep: select lines that match a pattern

- Usage: `grep pattern [file] [file] ...`
- No files specified: reads from standard input
  - Do you see a pattern yet?
- Simple pattern: plain text
- Fancier patterns: later in the semester
- Writes lines that contain the pattern

```
prompt$ ls
1.SpeakToMe.mp3     4.TheGreatGig.mp3  7.AnyColourYo.mp3
2.OnTheRun.mp3      5.Money.mp3        8.BrainDamage.mp3
3.Time.mp3          6.UsAndThem.mp3    9.Eclipse.mp3
prompt$ █
```

grep: select lines that match a pattern

- Usage: `grep pattern [file] [file] ...`
- No files specified: reads from standard input
  - Do you see a pattern yet?
- Simple pattern: plain text
- Fancier patterns: later in the semester
- Writes lines that contain the pattern

```
prompt$ ls
1.SpeakToMe.mp3    4.TheGreatGig.mp3  7.AnyColourYo.mp3
2.OnTheRun.mp3     5.Money.mp3        8.BrainDamage.mp3
3.Time.mp3         6.UsAndThem.mp3    9.Eclipse.mp3
prompt$ ls | grep The
```

grep: select lines that match a pattern

- Usage: `grep pattern [file] [file] ...`
- No files specified: reads from standard input
  - Do you see a pattern yet?
- Simple pattern: plain text
- Fancier patterns: later in the semester
- Writes lines that contain the pattern

```
prompt$ ls
1.SpeakToMe.mp3    4.TheGreatGig.mp3  7.AnyColourYo.mp3
2.OnTheRun.mp3     5.Money.mp3        8.BrainDamage.mp3
3.Time.mp3         6.UsAndThem.mp3    9.Eclipse.mp3
prompt$ ls | grep The
2.OnTheRun.mp3
4.TheGreatGig.mp3
6.UsAndThem.mp3
prompt$ 
```

grep tricks

- grep -v: Invert the pattern
  - Writes lines that do not contain the pattern
- Using grep with more than one file
  - The filename is shown ahead of each matching line
  - Useful, say, to find “which file did I write. . .”

```
prompt$ █
```

grep tricks

- `grep -v`: Invert the pattern
  - Writes lines that do not contain the pattern
- Using `grep` with more than one file
  - The filename is shown ahead of each matching line
  - Useful, say, to find "which file did I write..."

```
prompt$ grep "forest()" *.h
```

grep tricks

- `grep -v`: Invert the pattern
  - Writes lines that do not contain the pattern
- Using `grep` with more than one file
  - The filename is shown ahead of each matching line
  - Useful, say, to find "which file did I write..."

```
prompt$ grep "forest()" *.h
meddly.h:    virtual ~forest();
super.h:    virtual ~super_forest();
prompt$ 
```

cat: concatenate files

- Usage: `cat [file] [file] ...`
- Writes files to standard output, in order
- How can we use this as a “filter”?

`cat`: concatenate files

- Usage: `cat [file] [file] ...`
- Writes files to standard output, in order
- How can we use this as a "filter"?
    1. If no files are specified, reads from standard input

cat: concatenate files

- Usage: `cat [file] [file] ...`
- Writes files to standard output, in order
- How can we use this as a "filter"?
  1. If no files are specified, reads from standard input
  2. If a file name "–" is given, reads from standard input
     (This only works once)

cat: concatenate files

- Usage: `cat [file] [file] ...`
- Writes files to standard output, in order
- How can we use this as a "filter"?
    1. If no files are specified, reads from standard input
    2. If a file name "-" is given, reads from standard input
       (This only works once)
- Why are these useful?

cat: concatenate files

- Usage: `cat [file] [file] ...`
- Writes files to standard output, in order
- How can we use this as a "filter"?
  1. If no files are specified, reads from standard input
  2. If a file name "-" is given, reads from standard input (This only works once)
- Why are these useful?
  1. cat has some useful switches

`-n`	: number lines
`-s`	: squeeze several blank lines into one

`cat`: concatenate files

- Usage: `cat [file] [file] ...`
- Writes files to standard output, in order
- How can we use this as a "filter"?
    1. If no files are specified, reads from standard input
    2. If a file name "-" is given, reads from standard input (This only works once)
- Why are these useful?
    1. cat has some useful switches

`-n`	: number lines
`-s`	: squeeze several blank lines into one

    2. Append things to standard input

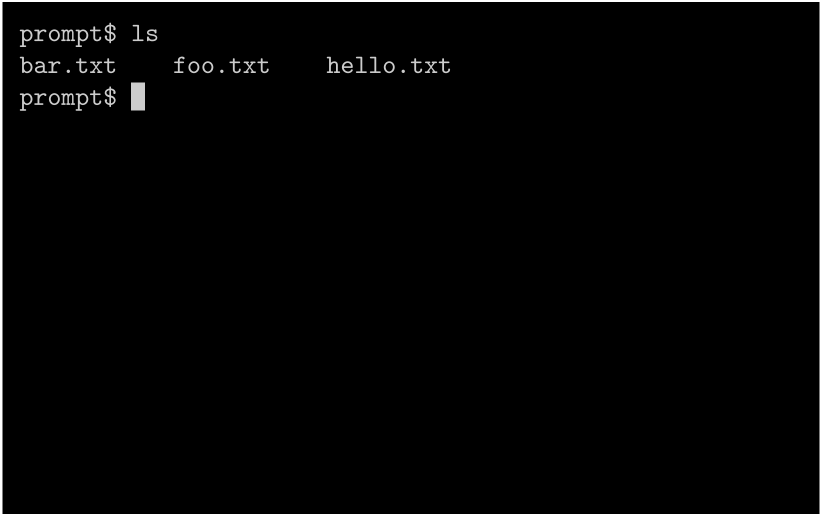
cat examples

```
prompt$ █
```

cat examples

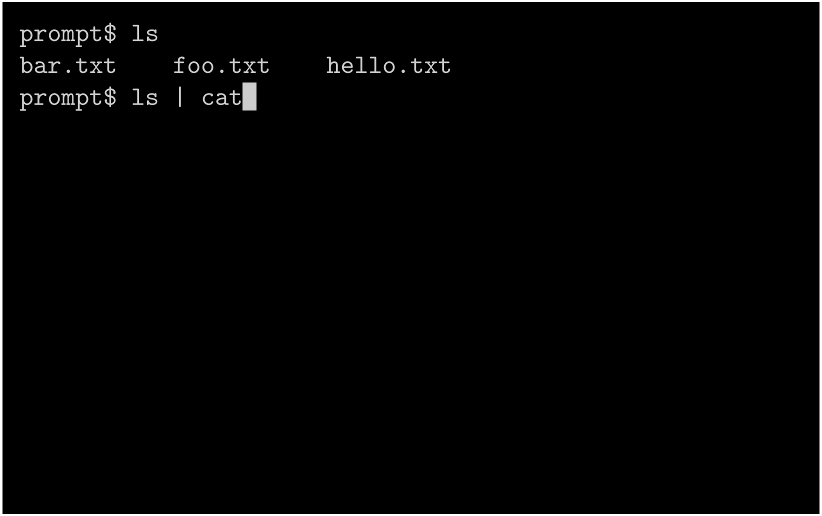
```
prompt$ ls
```

cat examples



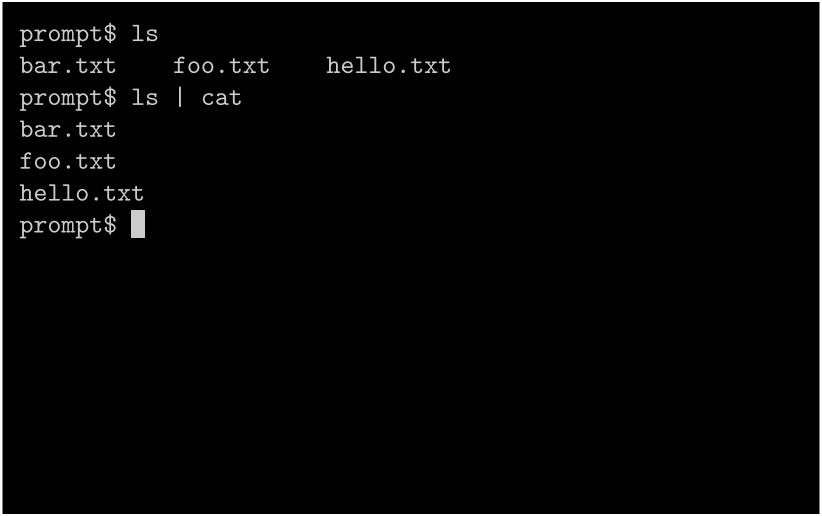
```
prompt$ ls
bar.txt    foo.txt    hello.txt
prompt$ 
```

cat examples



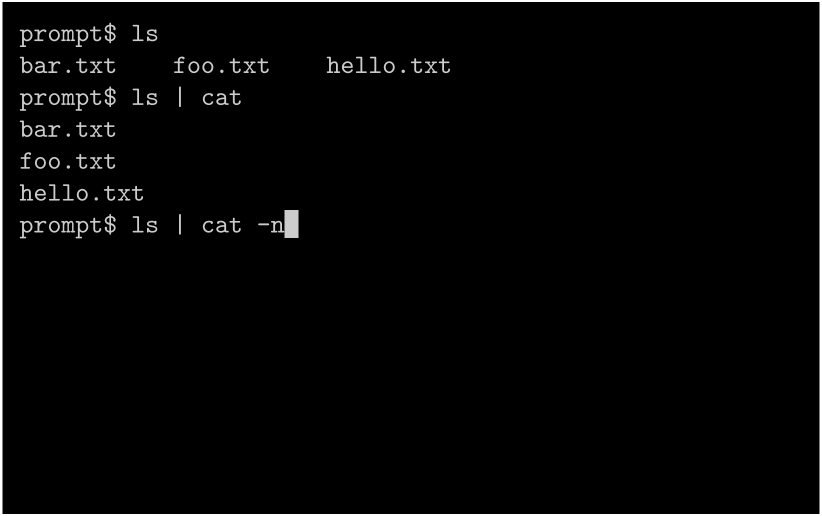
```
prompt$ ls
bar.txt     foo.txt     hello.txt
prompt$ ls | cat
```

cat examples



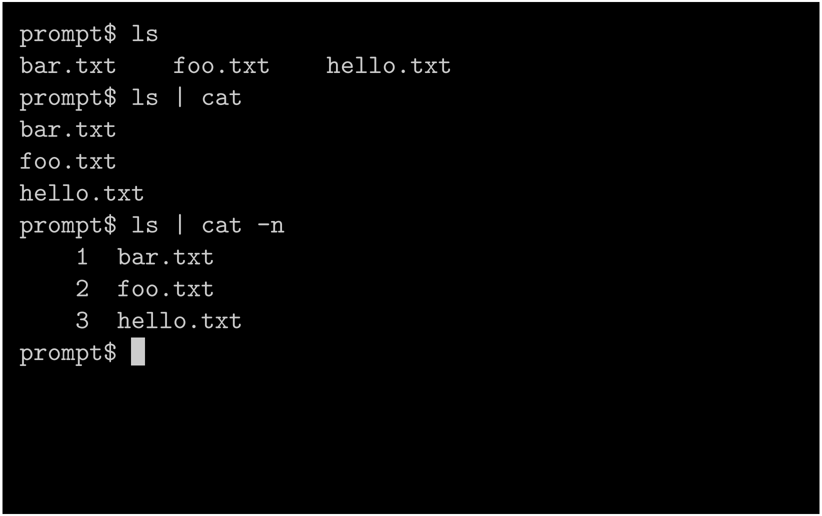
```
prompt$ ls
bar.txt    foo.txt    hello.txt
prompt$ ls | cat
bar.txt
foo.txt
hello.txt
prompt$ 
```

cat examples



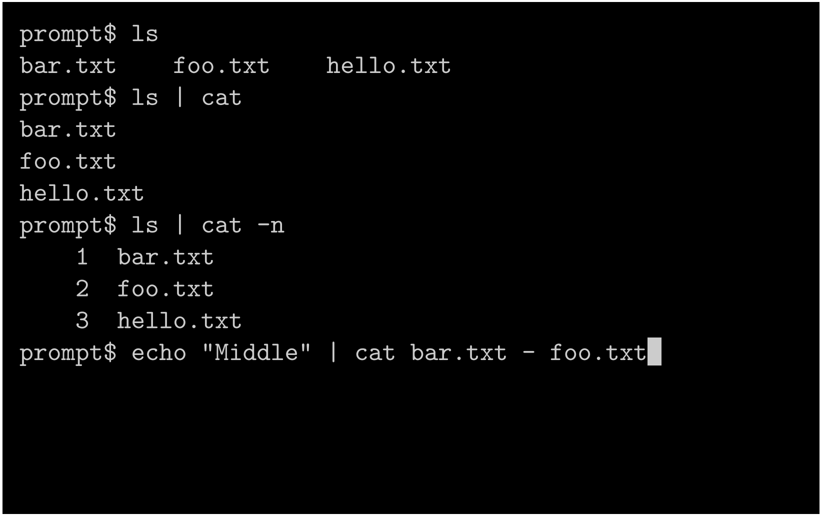
```
prompt$ ls
bar.txt    foo.txt    hello.txt
prompt$ ls | cat
bar.txt
foo.txt
hello.txt
prompt$ ls | cat -n
```

cat examples



```
prompt$ ls
bar.txt    foo.txt    hello.txt
prompt$ ls | cat
bar.txt
foo.txt
hello.txt
prompt$ ls | cat -n
     1  bar.txt
     2  foo.txt
     3  hello.txt
prompt$ 
```

cat examples



```
prompt$ ls
bar.txt    foo.txt    hello.txt
prompt$ ls | cat
bar.txt
foo.txt
hello.txt
prompt$ ls | cat -n
     1  bar.txt
     2  foo.txt
     3  hello.txt
prompt$ echo "Middle" | cat bar.txt - foo.txt
```

cat examples

```
prompt$ ls
bar.txt    foo.txt    hello.txt
prompt$ ls | cat
bar.txt
foo.txt
hello.txt
prompt$ ls | cat -n
    1  bar.txt
    2  foo.txt
    3  hello.txt
prompt$ echo "Middle" | cat bar.txt - foo.txt
This is file bar.txt
Middle
This is file foo.txt
prompt$ 
```

cat tricks

What happens if I just do:

```
prompt$ cat
```

cat tricks

What happens if I just do:

```
prompt$ cat
```

- Read from standard input

cat tricks

What happens if I just do:

```
prompt$ cat
```

- Read from standard input
- Write to standard output

cat tricks

What happens if I just do:

```
prompt$ cat
```

- Read from standard input
- Write to standard output

Is this useful?

cat tricks

What happens if I just do:

```
prompt$ cat
```

- Read from standard input
- Write to standard output

Is this useful?

- If all else fails, I can create a text file

  ```
  prompt$ cat > file.txt
  ```

  Whatever I type, goes into the file

cat tricks

What happens if I just do:

```
prompt$ cat
```

- Read from standard input
- Write to standard output

Is this useful?

- If all else fails, I can create a text file

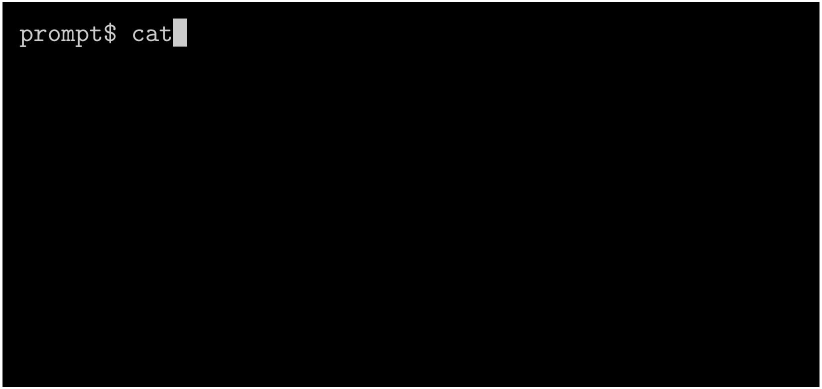
  ```
  prompt$ cat > file.txt
  ```

  Whatever I type, goes into the file
- Use `Ctrl-D` to indicate end of file

cat trick examples

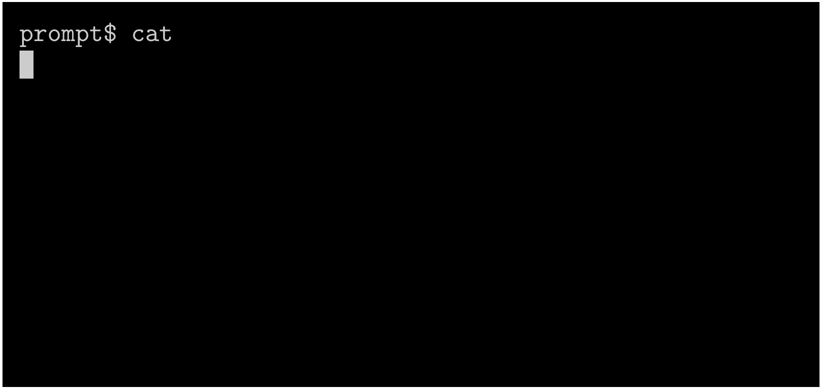
```
prompt$ █
```

cat trick examples



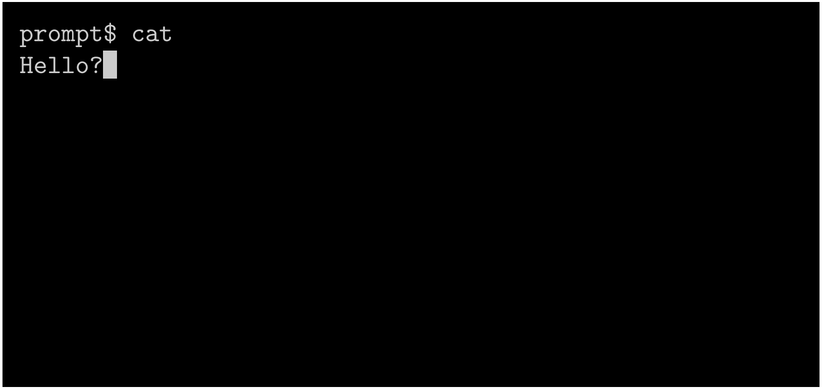
```
prompt$ cat
```

cat trick examples



```
prompt$ cat
█
```

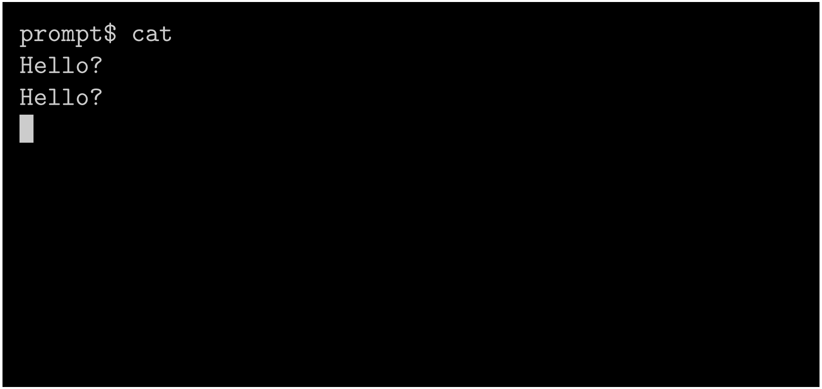
cat trick examples



```
prompt$ cat
Hello?
```

(I typed this)

cat trick examples



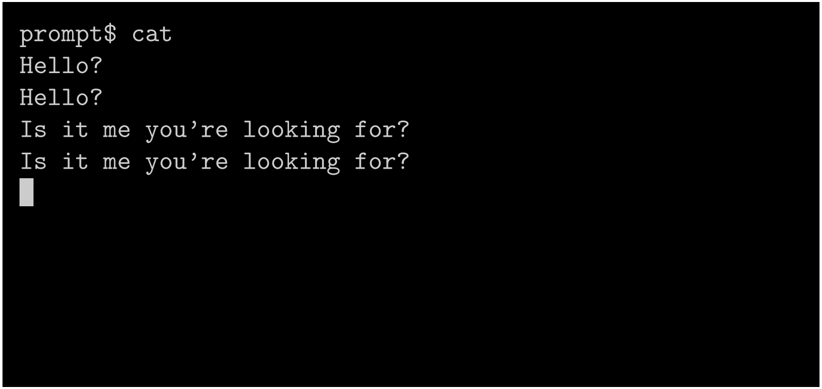
```
prompt$ cat
Hello?
Hello?
█
```

cat trick examples

```
prompt$ cat
Hello?
Hello?
Is it me you're looking for?
```

(I typed this)

cat trick examples



```
prompt$ cat
Hello?
Hello?
Is it me you're looking for?
Is it me you're looking for?
█
```

(Hit Ctrl-D)

cat trick examples

```
prompt$ cat
Hello?
Hello?
Is it me you're looking for?
Is it me you're looking for?
prompt$ █
```

cat trick examples

```
prompt$ cat
Hello?
Hello?
Is it me you're looking for?
Is it me you're looking for?
prompt$ cat > catfile.txt
```

cat trick examples

```
prompt$ cat
Hello?
Hello?
Is it me you're looking for?
Is it me you're looking for?
prompt$ cat > catfile.txt
█
```

cat trick examples

```
prompt$ cat
Hello?
Hello?
Is it me you’re looking for?
Is it me you’re looking for?
prompt$ cat > catfile.txt
I’m making this file
```

(I am typing this)

cat trick examples

```
prompt$ cat
Hello?
Hello?
Is it me you're looking for?
Is it me you're looking for?
prompt$ cat > catfile.txt
I'm making this file

```

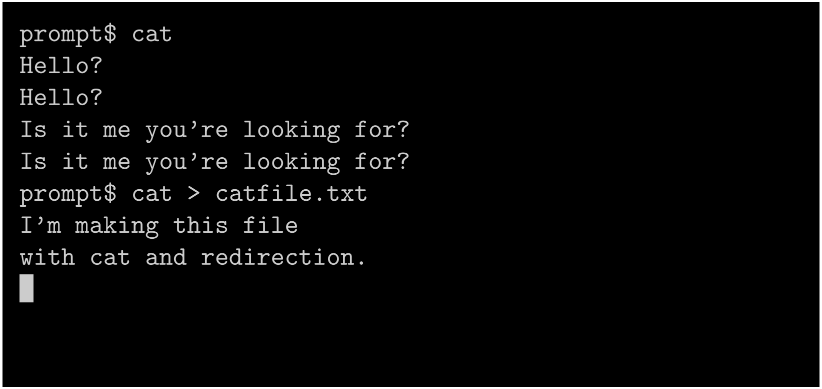
(I am typing this)

cat trick examples

```
prompt$ cat
Hello?
Hello?
Is it me you're looking for?
Is it me you're looking for?
prompt$ cat > catfile.txt
I'm making this file
with cat and redirection.
```

(I am typing this)

cat trick examples



```
prompt$ cat
Hello?
Hello?
Is it me you're looking for?
Is it me you're looking for?
prompt$ cat > catfile.txt
I'm making this file
with cat and redirection.
█
```

(I am typing this)

cat trick examples

```
prompt$ cat
Hello?
Hello?
Is it me you're looking for?
Is it me you're looking for?
prompt$ cat > catfile.txt
I'm making this file
with cat and redirection.
An editor would be better.
```

(I am typing this)

cat trick examples

```
prompt$ cat
Hello?
Hello?
Is it me you're looking for?
Is it me you're looking for?
prompt$ cat > catfile.txt
I'm making this file
with cat and redirection.
An editor would be better.
```

(I am typing this)

cat trick examples

```
prompt$ cat
Hello?
Hello?
Is it me you're looking for?
Is it me you're looking for?
prompt$ cat > catfile.txt
I'm making this file
with cat and redirection.
An editor would be better.
But it is better than using echo.
```

(I am typing this)

cat trick examples

```
prompt$ cat
Hello?
Hello?
Is it me you're looking for?
Is it me you're looking for?
prompt$ cat > catfile.txt
I'm making this file
with cat and redirection.
An editor would be better.
But it is better than using echo.
█
```

(Hit Ctrl-D)

cat trick examples

```
prompt$ cat
Hello?
Hello?
Is it me you're looking for?
Is it me you're looking for?
prompt$ cat > catfile.txt
I'm making this file
with cat and redirection.
An editor would be better.
But it is better than using echo.
prompt$
```

`sort`: sort a file

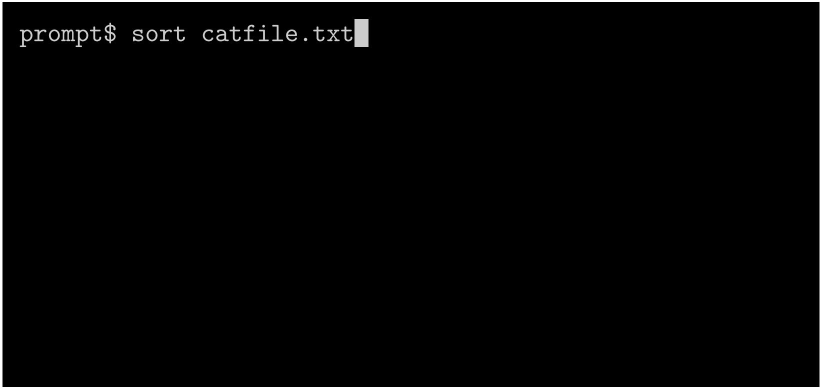
- Usage: `sort [file] [file]`
  - Concatenates files and sorts them
  - No file specified: read from standard input
- Lots of useful options, check your `man` pages

`-k`	key position (default: 1) (column to sort by)
`-n`	numeric sort (otherwise – alphabetical sort)
`-r`	reverse sort
`-u`	merge unique

sort examples

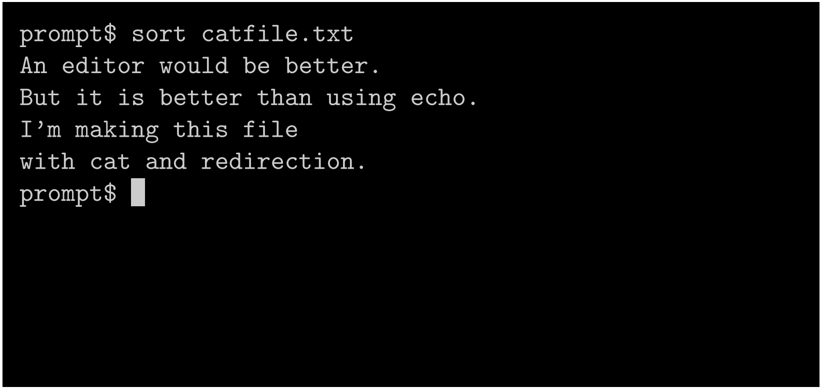
```
prompt$ 
```

sort examples



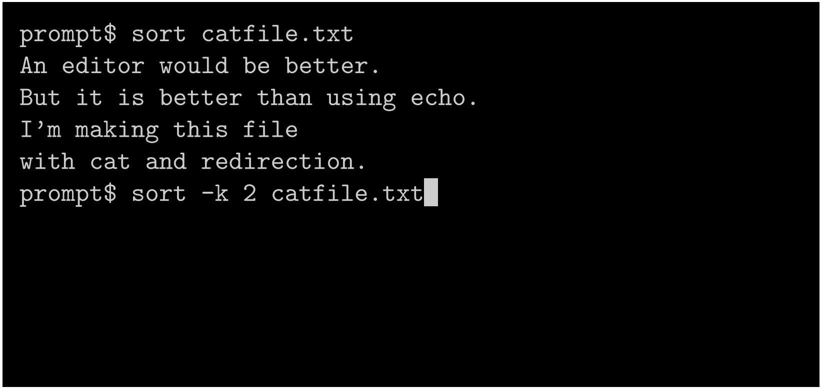
```
prompt$ sort catfile.txt
```

sort examples



```
prompt$ sort catfile.txt
An editor would be better.
But it is better than using echo.
I'm making this file
with cat and redirection.
prompt$ 
```

sort examples



```
prompt$ sort catfile.txt
An editor would be better.
But it is better than using echo.
I'm making this file
with cat and redirection.
prompt$ sort -k 2 catfile.txt
```

sort examples

```
prompt$ sort catfile.txt
An editor would be better.
But it is better than using echo.
I'm making this file
with cat and redirection.
prompt$ sort -k 2 catfile.txt
with cat and redirection.
An editor would be better.
But it is better than using echo.
I'm making this file
prompt$
```

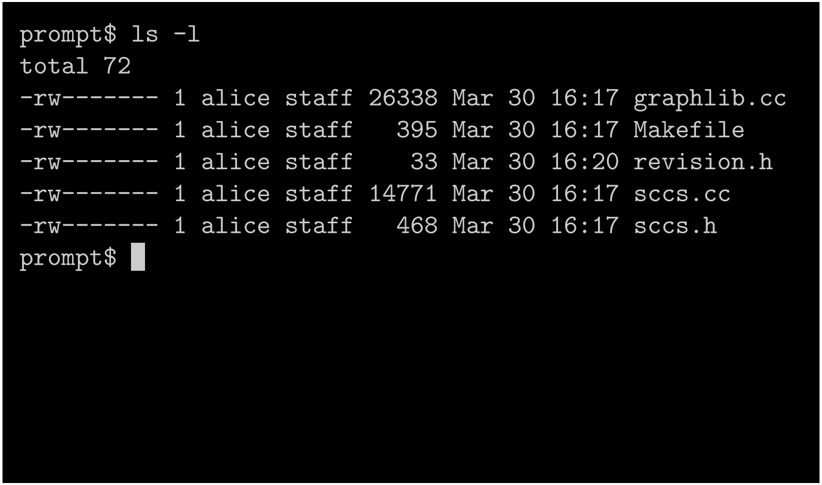
sort examples (2)

```
prompt$
```

sort examples (2)

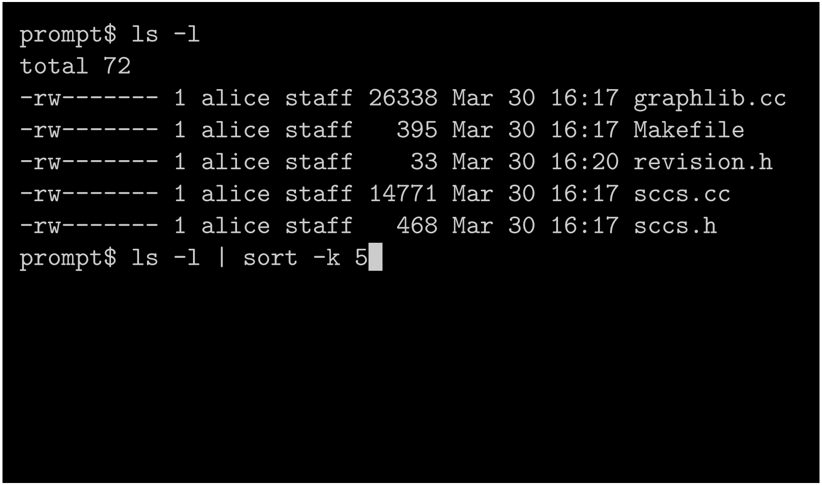
```
prompt$ ls -l
```

sort examples (2)



```
prompt$ ls -l
total 72
-rw------- 1 alice staff 26338 Mar 30 16:17 graphlib.cc
-rw------- 1 alice staff   395 Mar 30 16:17 Makefile
-rw------- 1 alice staff    33 Mar 30 16:20 revision.h
-rw------- 1 alice staff 14771 Mar 30 16:17 sccs.cc
-rw------- 1 alice staff   468 Mar 30 16:17 sccs.h
prompt$
```

sort examples (2)



```
prompt$ ls -l
total 72
-rw------- 1 alice staff 26338 Mar 30 16:17 graphlib.cc
-rw------- 1 alice staff   395 Mar 30 16:17 Makefile
-rw------- 1 alice staff    33 Mar 30 16:20 revision.h
-rw------- 1 alice staff 14771 Mar 30 16:17 sccs.cc
-rw------- 1 alice staff   468 Mar 30 16:17 sccs.h
prompt$ ls -l | sort -k 5
```

sort examples (2)

```
total 72
-rw------- 1 alice staff 26338 Mar 30 16:17 graphlib.cc
-rw------- 1 alice staff   395 Mar 30 16:17 Makefile
-rw------- 1 alice staff    33 Mar 30 16:20 revision.h
-rw------- 1 alice staff 14771 Mar 30 16:17 sccs.cc
-rw------- 1 alice staff   468 Mar 30 16:17 sccs.h
prompt$ ls -l | sort -k 5
total 72
-rw------- 1 alice staff 14771 Mar 30 16:17 sccs.cc
-rw------- 1 alice staff 26338 Mar 30 16:17 graphlib.cc
-rw------- 1 alice staff    33 Mar 30 16:20 revision.h
-rw------- 1 alice staff   395 Mar 30 16:17 Makefile
-rw------- 1 alice staff   468 Mar 30 16:17 sccs.h
prompt$ 
```

sort examples (2)

```
total 72
-rw------- 1 alice staff 26338 Mar 30 16:17 graphlib.cc
-rw------- 1 alice staff   395 Mar 30 16:17 Makefile
-rw------- 1 alice staff    33 Mar 30 16:20 revision.h
-rw------- 1 alice staff 14771 Mar 30 16:17 sccs.cc
-rw------- 1 alice staff   468 Mar 30 16:17 sccs.h
prompt$ ls -l | sort -k 5
total 72
-rw------- 1 alice staff 14771 Mar 30 16:17 sccs.cc
-rw------- 1 alice staff 26338 Mar 30 16:17 graphlib.cc
-rw------- 1 alice staff    33 Mar 30 16:20 revision.h
-rw------- 1 alice staff   395 Mar 30 16:17 Makefile
-rw------- 1 alice staff   468 Mar 30 16:17 sccs.h
prompt$ !! -n
```

sort examples (2)

```
-rw------- 1 alice staff 14771 Mar 30 16:17 sccs.cc
-rw------- 1 alice staff 26338 Mar 30 16:17 graphlib.cc
-rw------- 1 alice staff    33 Mar 30 16:20 revision.h
-rw------- 1 alice staff   395 Mar 30 16:17 Makefile
-rw------- 1 alice staff   468 Mar 30 16:17 sccs.h
prompt$ !! -n
ls -l | sort -k 5 -n
total 72
-rw------- 1 alice staff    33 Mar 30 16:20 revision.h
-rw------- 1 alice staff   395 Mar 30 16:17 Makefile
-rw------- 1 alice staff   468 Mar 30 16:17 sccs.h
-rw------- 1 alice staff 14771 Mar 30 16:17 sccs.cc
-rw------- 1 alice staff 26338 Mar 30 16:17 graphlib.cc
prompt$ 
```

sort examples (2)

```
-rw------- 1 alice staff 14771 Mar 30 16:17 sccs.cc
-rw------- 1 alice staff 26338 Mar 30 16:17 graphlib.cc
-rw------- 1 alice staff    33 Mar 30 16:20 revision.h
-rw------- 1 alice staff   395 Mar 30 16:17 Makefile
-rw------- 1 alice staff   468 Mar 30 16:17 sccs.h
prompt$ !! -n
ls -l | sort -k 5 -n
total 72
-rw------- 1 alice staff    33 Mar 30 16:20 revision.h
-rw------- 1 alice staff   395 Mar 30 16:17 Makefile
-rw------- 1 alice staff   468 Mar 30 16:17 sccs.h
-rw------- 1 alice staff 14771 Mar 30 16:17 sccs.cc
-rw------- 1 alice staff 26338 Mar 30 16:17 graphlib.cc
prompt$
```

(Turns out: `ls -lSr` does the same thing . . .)

`tr`: translate characters

- Copies standard input to standard output
- Some characters are changed
- Usage 1: `tr string1 string2`
  - `string1` and `string2` are lists of characters
  - If standard input character appears at position *n* of `string1` then write character at position *n* of `string2`
  - All other characters are copied with no change
- Usage 2: `tr -d string`
  - Any character appearing in `string` is not copied
  - All other characters are copied with no change
- There are other uses. Check your `man` pages.

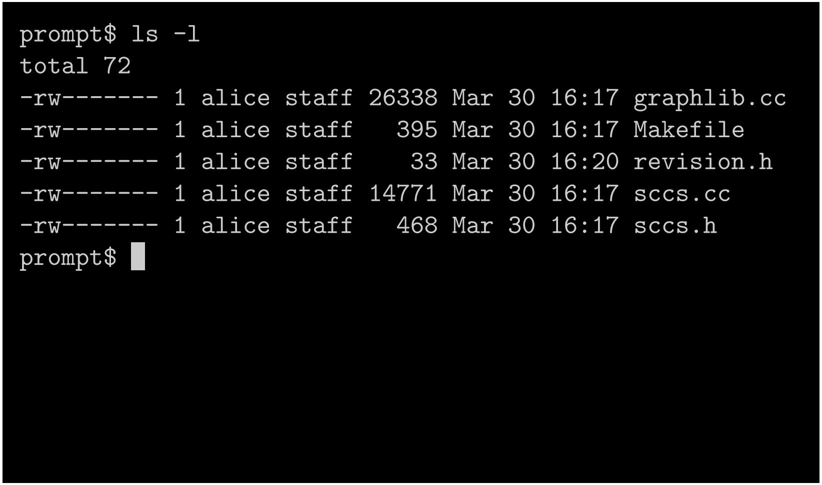
tr examples

```
prompt$
```

tr examples

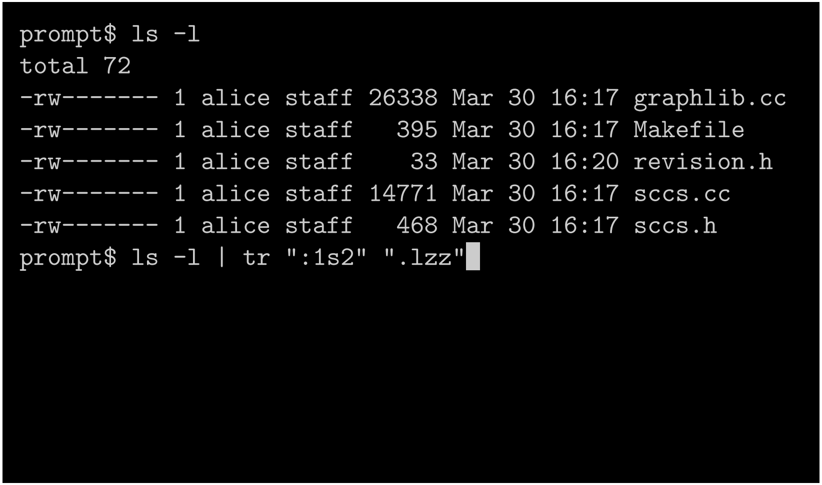
```
prompt$ ls -l
```

tr examples



```
prompt$ ls -l
total 72
-rw------- 1 alice staff 26338 Mar 30 16:17 graphlib.cc
-rw------- 1 alice staff   395 Mar 30 16:17 Makefile
-rw------- 1 alice staff    33 Mar 30 16:20 revision.h
-rw------- 1 alice staff 14771 Mar 30 16:17 sccs.cc
-rw------- 1 alice staff   468 Mar 30 16:17 sccs.h
prompt$ 
```

tr examples



```
prompt$ ls -l
total 72
-rw------- 1 alice staff 26338 Mar 30 16:17 graphlib.cc
-rw------- 1 alice staff   395 Mar 30 16:17 Makefile
-rw------- 1 alice staff    33 Mar 30 16:20 revision.h
-rw------- 1 alice staff 14771 Mar 30 16:17 sccs.cc
-rw------- 1 alice staff   468 Mar 30 16:17 sccs.h
prompt$ ls -l | tr ":1s2" ".lzz"
```

tr examples

```
total 72
-rw------- 1 alice staff 26338 Mar 30 16:17 graphlib.cc
-rw------- 1 alice staff   395 Mar 30 16:17 Makefile
-rw------- 1 alice staff    33 Mar 30 16:20 revision.h
-rw------- 1 alice staff 14771 Mar 30 16:17 sccs.cc
-rw------- 1 alice staff   468 Mar 30 16:17 sccs.h
prompt$ ls -l | tr ":1s2" ".lzz"
total 7z
-rw------- l alice ztaff z6338 Mar 30 l6.l7 graphlib.cc
-rw------- l alice ztaff   395 Mar 30 l6.l7 Makefile
-rw------- l alice ztaff    33 Mar 30 l6.z0 revizion.h
-rw------- l alice ztaff l477l Mar 30 l6.l7 zccz.cc
-rw------- l alice ztaff   468 Mar 30 l6.l7 zccz.h
prompt$ █
```

tr examples

```
total 72
-rw------- 1 alice staff 26338 Mar 30 16:17 graphlib.cc
-rw------- 1 alice staff   395 Mar 30 16:17 Makefile
-rw------- 1 alice staff    33 Mar 30 16:20 revision.h
-rw------- 1 alice staff 14771 Mar 30 16:17 sccs.cc
-rw------- 1 alice staff   468 Mar 30 16:17 sccs.h
prompt$ ls -l | tr ":1s2" ".lzz"
total 7z
-rw------- l alice ztaff z6338 Mar 30 l6.l7 graphlib.cc
-rw------- l alice ztaff   395 Mar 30 l6.l7 Makefile
-rw------- l alice ztaff    33 Mar 30 l6.z0 revizion.h
-rw------- l alice ztaff l477l Mar 30 l6.l7 zccz.cc
-rw------- l alice ztaff   468 Mar 30 l6.l7 zccz.h
prompt$ ls -l | tr -d aeiou
```

tr examples

```
total 7z
-rw------- 1 alice ztaff z6338 Mar 30 16.17 graphlib.cc
-rw------- 1 alice ztaff   395 Mar 30 16.17 Makefile
-rw------- 1 alice ztaff    33 Mar 30 16.z0 revizion.h
-rw------- 1 alice ztaff 14771 Mar 30 16.17 zccz.cc
-rw------- 1 alice ztaff   468 Mar 30 16.17 zccz.h
prompt$ ls -l | tr -d aeiou
ttl 72
-rw------- 1 lc stff 26338 Mr 30 16:17 grphlb.cc
-rw------- 1 lc stff   395 Mr 30 16:17 Mkfl
-rw------- 1 lc stff    33 Mr 30 16:20 rvsn.h
-rw------- 1 lc stff 14771 Mr 30 16:17 sccs.cc
-rw------- 1 lc stff   468 Mr 30 16:17 sccs.h
prompt$ █
```

tee: pipeline splitter

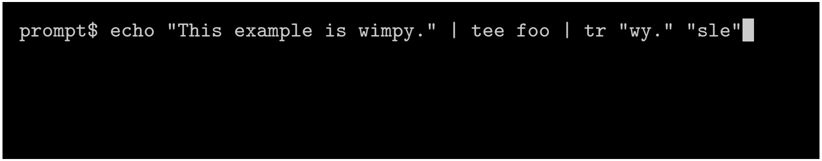
- Usage: `tee [file] [file] ...`
- Copies standard input to standard output
- . . . and to all the files specified
- Think of pipe fitting in the shape of a “T”
- Has options to append files instead of overwriting them
  - “Consult your `man` pages”[1]

```
prompt$ 
```

[1]That’s my new catch phrase

tee: pipeline splitter

- Usage: `tee [file] [file] ...`
- Copies standard input to standard output
- ...and to all the files specified
- Think of pipe fitting in the shape of a "T"
- Has options to append files instead of overwriting them
  - "Consult your `man` pages"[1]



```
prompt$ echo "This example is wimpy." | tee foo | tr "wy." "sle"
```

[1]That's my new catch phrase

tee: pipeline splitter

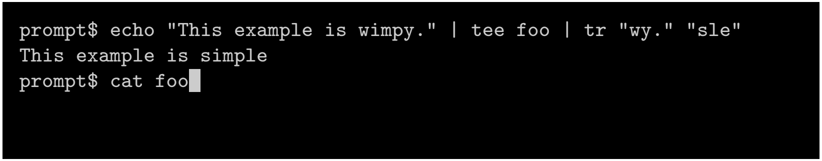
- Usage: `tee [file] [file] ...`
- Copies standard input to standard output
- . . . and to all the files specified
- Think of pipe fitting in the shape of a "T"
- Has options to append files instead of overwriting them
  - "Consult your `man` pages"[1]

```
prompt$ echo "This example is wimpy." | tee foo | tr "wy." "sle"
This example is simple
prompt$ 
```

[1] That's my new catch phrase

tee: pipeline splitter

- Usage: `tee [file] [file] ...`
- Copies standard input to standard output
- ...and to all the files specified
- Think of pipe fitting in the shape of a "T"
- Has options to append files instead of overwriting them
  - "Consult your `man` pages"[1]



```
prompt$ echo "This example is wimpy." | tee foo | tr "wy." "sle"
This example is simple
prompt$ cat foo
```

[1] That's my new catch phrase

tee: pipeline splitter

- Usage: `tee [file] [file] ...`
- Copies standard input to standard output
- ...and to all the files specified
- Think of pipe fitting in the shape of a “T”
- Has options to append files instead of overwriting them
  - “Consult your `man` pages”[1]

```
prompt$ echo "This example is wimpy." | tee foo | tr "wy." "sle"
This example is simple
prompt$ cat foo
This example is wimpy.
prompt$ 
```

[1] That’s my new catch phrase

Time for a quiz

- Suppose we are working on an open source project
- Each source file has the same few paragraphs at the top
  - Some brief license information
- These paragraphs are stored in a file named `blank`
- Want to create six new source files
- Question: how can I make six copies of file `blank`?
  - With nothing displayed on my terminal
  - In the fewest possible keystrokes, because I'm lazy

Time for a quiz

- Suppose we are working on an open source project
- Each source file has the same few paragraphs at the top
  - Some brief license information
- These paragraphs are stored in a file named `blank`
- Want to create six new source files
- Question: how can I make six copies of file `blank`?
  - With nothing displayed on my terminal
  - In the fewest possible keystrokes, because I'm lazy

```
prompt$ tee file1 file2 file3 file4 file5 > file6 < blank
```

Strange but useful utilities

`wc`: count words, lines of a file

- Usage: `wc [file] [file] ...`
- Display the number of lines, words, and characters in each file
- If more than one file, also display the total
- If no files specified — read from standard input
- Can use switches to get just one value (e.g., just # lines)
  - "Consult your `man` pages" as usual

```
prompt$ █
```

Strange but useful utilities

wc: count words, lines of a file

- Usage: `wc [file] [file] ...`
- Display the number of lines, words, and characters in each file
- If more than one file, also display the total
- If no files specified — read from standard input
- Can use switches to get just one value (e.g., just # lines)
  - "Consult your `man` pages" as usual

```
prompt$ wc catfile.txt
```

Strange but useful utilities

wc: count words, lines of a file

- Usage: `wc [file] [file] ...`
- Display the number of lines, words, and characters in each file
- If more than one file, also display the total
- If no files specified — read from standard input
- Can use switches to get just one value (e.g., just # lines)
  - "Consult your `man` pages" as usual

```
prompt$ wc catfile.txt
       4       20      108 catfile.txt
prompt$ 
```

Strange but useful utilities

wc: count words, lines of a file

- Usage: `wc [file] [file] ...`
- Display the number of lines, words, and characters in each file
- If more than one file, also display the total
- If no files specified — read from standard input
- Can use switches to get just one value (e.g., just # lines)
  - "Consult your `man` pages" as usual

```
prompt$ wc catfile.txt
       4       20      108 catfile.txt
prompt$ ls | wc
```

Strange but useful utilities

wc: count words, lines of a file

- Usage: `wc [file] [file] ...`
- Display the number of lines, words, and characters in each file
- If more than one file, also display the total
- If no files specified — read from standard input
- Can use switches to get just one value (e.g., just # lines)
  - "Consult your `man` pages" as usual

```
prompt$ wc catfile.txt
       4       20      108 catfile.txt
prompt$ ls | wc
      22       22      215
prompt$ █
```

yes: write something, forever

- Usage: `yes [expletive]`
- Writes `expletive` to standard output, forever
- If no expletive is given, default is "y"

```
prompt$ █
```

yes: write something, forever

- Usage: `yes [expletive]`
- Writes `expletive` to standard output, forever
- If no expletive is given, default is "y"

```
prompt$ yes no
```

yes: write something, forever

- Usage: `yes [expletive]`
- Writes `expletive` to standard output, forever
- If no expletive is given, default is "y"

```
no
no
no
no
no
no
no
```

This is scrolling and printing more. . .

yes: write something, forever

- Usage: `yes [expletive]`
- Writes `expletive` to standard output, forever
- If no expletive is given, default is “y”

```
no
no
no
no
no
^C
prompt$ █
```

What is a good use for yes?

What is a good use for yes?

- Suppose I run a utility that asks me lots of yes/no questions
- Suppose I know that every question will be answered "y"
- Then if I type

```
prompt$ yes | utility
```

all questions will be answered "y"

Realistic yes example

- Suppose your shell is set up so that
  `rm` prompts before removing each file
  - In a few lectures, we will see how to set this up
- To remove directory `Foo` and all its files:

```
prompt$ █
```

Realistic yes example

- Suppose your shell is set up so that `rm` prompts before removing each file
  - In a few lectures, we will see how to set this up
- To remove directory `Foo` and all its files:

```
prompt$ yes | rm -R Foo
```

Realistic yes example

- Suppose your shell is set up so that `rm` prompts before removing each file
  - In a few lectures, we will see how to set this up
- To remove directory `Foo` and all its files:

```
prompt$ yes | rm -R Foo
rm: descend into directory ‘Foo’? rm: remove regular file
 ‘Foo/bar.cc’? rm: remove regular file ‘Foo/bar.h’? rm: r
emove regular file Foo/a.out? rm: remove regular file ‘Fo
o/hello.c’? rm: remove regular file ‘Foo/Makefile’? rm: r
emove regular file Foo/core?  prompt$
```

`xargs`: extract arguments from standard input

- Usage: `xargs [utility [arguments]]`
- Reads from standard input
- Grabs words (strings) from standard input
  - Words are separated by "whitespace": spaces, tabs, and new lines
- Passes words as arguments to `utility`
  - Default utility if none specified: `echo`
- Lots of useful switches and options
  - Check your `man` pages for details

Generic example for xargs

```
prompt$ cat wordfile
This is a simple
text file with a few
words
prompt$
```

Generic example for xargs

```
prompt$ cat wordfile
This is a simple
text file with a few
words
prompt$
```

If we do this:

```
prompt$ cat wordfile | xargs utility arg1 arg2
```

Generic example for xargs

```
prompt$ cat wordfile
This is a simple
text file with a few
words
prompt$
```

If we do this:

```
prompt$ cat wordfile | xargs utility arg1 arg2
```

Then `xargs` splits standard input into words and each word becomes an argument:

```
utility arg1 arg2 This is a simple text file with a few words
```

Tricks with xargs

Limiting the number of arguments

- A long input stream can produce *lots* of arguments
- . . . Maybe too many (there is a limit)

-n : Allows us to specify a limit for the number of arguments

  - The utility may be executed several times

E.g., `cat wordfile | xargs -n 6 utility arg1 arg2` will execute:

```
utility arg1 arg2 This is a simple text file
utility arg1 arg2 with a few words
```

Tricks with xargs (2)

-I: specify a "replacement string"

- Each line from standard input is plugged in for the string
  - The `utility` may be executed several times

E.g., `cat wordfile | xargs -I % utility BEGIN % END` will execute:

```
utility BEGIN This is a simple END
utility BEGIN text file with a few END
utility BEGIN words END
```

Example killer apps for `xargs`

- Suppose I want to remove all files satisfying some criteria
  - E.g., "All files not owned by root, larger than 1 Gb in size"
- If I can come up with a pipeline to *list* those files . . .

Example killer apps for `xargs`

- Suppose I want to remove all files satisfying some criteria
  - E.g., "All files not owned by root, larger than 1 Gb in size"
- If I can come up with a pipeline to *list* those files ...
- ...then I can remove them with

```
prompt$ pipeline | to | list | files ...
```

Example killer apps for xargs

- Suppose I want to remove all files satisfying some criteria
  - E.g., "All files not owned by root, larger than 1 Gb in size"
- If I can come up with a pipeline to *list* those files . . .
- . . . then I can remove them with

```
prompt$ pipeline | to | list | files | xargs rm
```

Example killer apps for `xargs`

- Suppose I want to remove all files satisfying some criteria
  - E.g., "All files not owned by root, larger than 1 Gb in size"
- If I can come up with a pipeline to *list* those files . . .
- . . . then I can remove them with

```
prompt$ pipeline | to | list | files | xargs rm
```

- Or, I can terminate all processes satisfying some criteria

Example killer apps for xargs

- Suppose I want to remove all files satisfying some criteria
  - E.g., “All files not owned by root, larger than 1 Gb in size”
- If I can come up with a pipeline to *list* those files . . .
- . . . then I can remove them with

```
prompt$ pipeline | to | list | files | xargs rm
```

- Or, I can terminate all processes satisfying some criteria

```
prompt$ pipeline | to | list | PIDs | xargs kill
```

Some more serious examples

- Let's see how these simple utilities can be combined
- We will start with an easy one

Show lines 20 – 22 of `file.txt`

Show lines 20 – 22 of `file.txt`

`head` first (lines numbered for illustration)

```
prompt$
```

Show lines 20 – 22 of `file.txt`

head first (lines numbered for illustration)

```
prompt$ cat -n file.txt |
```

Show lines 20 – 22 of `file.txt`

head first (lines numbered for illustration)

```
prompt$ cat -n file.txt | head -n 22
```

Show lines 20 – 22 of `file.txt`

head first (lines numbered for illustration)

```
   18
   19  You will need g++, autoconf, automake and libtool.
   20
   21  0.  If you obtained the library via svn:
   22  $ ./autogen.sh
prompt$ 
```

Show lines 20 – 22 of `file.txt`

head first (lines numbered for illustration)

```
    18
    19  You will need g++, autoconf, automake and libtool.
    20
    21  0.  If you obtained the library via svn:
    22  $ ./autogen.sh
prompt$ cat -n file.txt | head -n 22 |
```

Show lines 20 – 22 of `file.txt`

head first (lines numbered for illustration)

```
    18
    19  You will need g++, autoconf, automake and libtool.
    20
    21  0.  If you obtained the library via svn:
    22  $ ./autogen.sh
prompt$ cat -n file.txt | head -n 22 | tail -n 3
```

Show lines 20 – 22 of `file.txt`

head first (lines numbered for illustration)

```
    22  $ ./autogen.sh
prompt$ cat -n file.txt | head -n 22 | tail -n 3
    20
    21  0.  If you obtained the library via svn:
    22  $ ./autogen.sh
prompt$ 
```

Show lines 20 – 22 of `file.txt`

head first (lines numbered for illustration)

```
   22  $ ./autogen.sh
prompt$ cat -n file.txt | head -n 22 | tail -n 3
   20
   21  0.  If you obtained the library via svn:
   22  $ ./autogen.sh
prompt$
```

tail first (lines numbered for illustration)

```
prompt$ █
```

Show lines 20 – 22 of `file.txt`

head first (lines numbered for illustration)

```
   22  $ ./autogen.sh
prompt$ cat -n file.txt | head -n 22 | tail -n 3
   20
   21  0.  If you obtained the library via svn:
   22  $ ./autogen.sh
prompt$
```

tail first (lines numbered for illustration)

```
prompt$ cat -n file.txt |
```

Show lines 20 – 22 of `file.txt`

head first (lines numbered for illustration)

```
    22  $ ./autogen.sh
prompt$ cat -n file.txt | head -n 22 | tail -n 3
    20
    21  0.  If you obtained the library via svn:
    22  $ ./autogen.sh
prompt$
```

tail first (lines numbered for illustration)

```
prompt$ cat -n file.txt | tail -n +20
```

Show lines 20 – 22 of `file.txt`

head first (lines numbered for illustration)

```
    22  $ ./autogen.sh
prompt$ cat -n file.txt | head -n 22 | tail -n 3
    20
    21  0.  If you obtained the library via svn:
    22  $ ./autogen.sh
prompt$
```

tail first (lines numbered for illustration)

```
prompt$ cat -n file.txt | tail -n +20
    20
    21  0.  If you obtained the library via svn:
    22  $ ./autogen.sh
    23  This will create the configure script and Makefiles.
    24
^C
prompt$
```

Show lines 20 – 22 of `file.txt`

head first (lines numbered for illustration)

```
   22  $ ./autogen.sh
prompt$ cat -n file.txt | head -n 22 | tail -n 3
   20
   21  0.  If you obtained the library via svn:
   22  $ ./autogen.sh
prompt$
```

tail first (lines numbered for illustration)

```
prompt$ cat -n file.txt | tail -n +20
   20
   21  0.  If you obtained the library via svn:
   22  $ ./autogen.sh
   23  This will create the configure script and Makefiles.
   24
^C
prompt$ cat -n file.txt | tail -n +20 |
```

Show lines 20 – 22 of `file.txt`

head first (lines numbered for illustration)

```
    22  $ ./autogen.sh
prompt$ cat -n file.txt | head -n 22 | tail -n 3
    20
    21  0.  If you obtained the library via svn:
    22  $ ./autogen.sh
prompt$
```

`tail` first (lines numbered for illustration)

```
prompt$ cat -n file.txt | tail -n +20
    20
    21  0.  If you obtained the library via svn:
    22  $ ./autogen.sh
    23  This will create the configure script and Makefiles.
    24
^C
prompt$ cat -n file.txt | tail -n +20 | head -n 3
```

Show lines 20 – 22 of `file.txt`

`head` first (lines numbered for illustration)

```
    22  $ ./autogen.sh
prompt$ cat -n file.txt | head -n 22 | tail -n 3
    20
    21  0.  If you obtained the library via svn:
    22  $ ./autogen.sh
prompt$
```

`tail` first (lines numbered for illustration)

```
    23  This will create the configure script and Makefiles.
    24
^C
prompt$ cat -n file.txt | tail -n +20 | head -n 3
    20
    21  0.  If you obtained the library via svn:
    22  $ ./autogen.sh
prompt$ 
```

Remove all files whose names contain “The”

Remove all files whose names contain “The”

First, how do I *list* files whose names contain “The”?

Remove all files whose names contain “The”

First, how do I *list* files whose names contain “The”?

```
prompt$ █
```

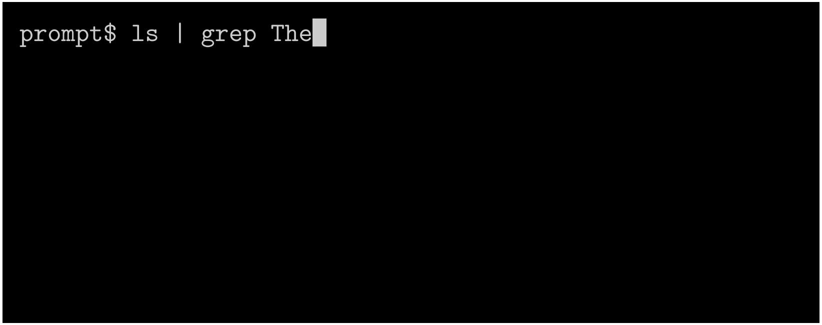
Remove all files whose names contain “The”

First, how do I *list* files whose names contain “The”?

```
prompt$ ls |
```

Remove all files whose names contain "The"

First, how do I *list* files whose names contain "The"?



```
prompt$ ls | grep The
```

Remove all files whose names contain “The”

First, how do I *list* files whose names contain “The”?

```
prompt$ ls | grep The
2.OnTheRun.mp3
4.TheGreatGig.mp3
6.UsAndThem.mp3
prompt$ █
```

Remove all files whose names contain “The”

Now, how to remove them?

```
prompt$ ls | grep The
2.OnTheRun.mp3
4.TheGreatGig.mp3
6.UsAndThem.mp3
prompt$ 
```

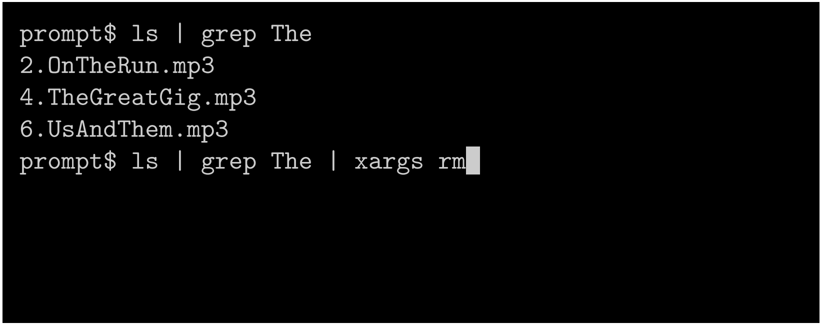
Remove all files whose names contain “The”

Now, how to remove them?

```
prompt$ ls | grep The
2.OnTheRun.mp3
4.TheGreatGig.mp3
6.UsAndThem.mp3
prompt$ ls | grep The |
```

Remove all files whose names contain “The”

Now, how to remove them?



```
prompt$ ls | grep The
2.OnTheRun.mp3
4.TheGreatGig.mp3
6.UsAndThem.mp3
prompt$ ls | grep The | xargs rm
```

Remove all files whose names contain “The”

Now, how to remove them?

```
prompt$ ls | grep The
2.OnTheRun.mp3
4.TheGreatGig.mp3
6.UsAndThem.mp3
prompt$ ls | grep The | xargs rm
prompt$ 
```

Remove all files whose names contain "The"

Now, how to remove them?

```
prompt$ ls | grep The
2.OnTheRun.mp3
4.TheGreatGig.mp3
6.UsAndThem.mp3
prompt$ ls | grep The | xargs rm
prompt$ ls
```

Remove all files whose names contain “The”

Now, how to remove them?

```
prompt$ ls | grep The
2.OnTheRun.mp3
4.TheGreatGig.mp3
6.UsAndThem.mp3
prompt$ ls | grep The | xargs rm
prompt$ ls
1.SpeakToMe.mp3    5.Money.mp3        8.BrainDamage.mp3
3.Time.mp3         7.AnyColourYo.mp3  9.Eclipse.mp3
prompt$ █
```

Now for a quiz

Now for a quiz

What does this do?

```
yes "" | cat -n | head -n 42 | xargs -n 1 echo +
  | xargs echo 0 | bc
```

Now for a quiz

What does this do?

```
yes "" | cat -n | head -n 42 | xargs -n 1 echo +
  | xargs echo 0 | bc
```

Hint: running this should produce

```
903
```

Answer to quiz

```
yes "" | cat -n | head -n 42 | xargs -n 1 echo +
  | xargs echo 0 | bc
```

`yes ""` :

`cat -n` :

`head -n 42` :

`xargs -n 1 echo +` :

`xargs echo 0` :

`bc` :

Answer to quiz

```
yes "" | cat -n | head -n 42 | xargs -n 1 echo +
  | xargs echo 0 | bc
```

`yes ""` : Print empty lines, forever

`cat -n` :

`head -n 42` :

`xargs -n 1 echo +` :

`xargs echo 0` :

`bc` :

Answer to quiz

```
yes "" | cat -n | head -n 42 | xargs -n 1 echo +
   | xargs echo 0 | bc
```

`yes ""` : Print empty lines, forever

`cat -n` : Number those empty lines

`head -n 42` :

`xargs -n 1 echo +` :

`xargs echo 0` :

`bc` :

Answer to quiz

```
yes "" | cat -n | head -n 42 | xargs -n 1 echo +
  | xargs echo 0 | bc
```

`yes ""`	Print empty lines, forever
`cat -n`	Number those empty lines
`head -n 42`	Stop after the first 42 lines
`xargs -n 1 echo +`	
`xargs echo 0`	
`bc`	

Answer to quiz

```
yes "" | cat -n | head -n 42 | xargs -n 1 echo +
  | xargs echo 0 | bc
```

`yes ""`	Print empty lines, forever
`cat -n`	Number those empty lines
`head -n 42`	Stop after the first 42 lines
`xargs -n 1 echo +`	Executes "`echo + word`" for each word in the input stream
`xargs echo 0`	
`bc`	

Answer to quiz

```
yes "" | cat -n | head -n 42 | xargs -n 1 echo +
   | xargs echo 0 | bc
```

`yes ""`	Print empty lines, forever
`cat -n`	Number those empty lines
`head -n 42`	Stop after the first 42 lines
`xargs -n 1 echo +`	Executes "`echo + word`" for each word in the input stream
`xargs echo 0`	Executes "`echo 0 all input stream words`"
`bc`	

Answer to quiz

```
yes "" | cat -n | head -n 42 | xargs -n 1 echo +
  | xargs echo 0 | bc
```

`yes ""`	Print empty lines, forever
`cat -n`	Number those empty lines
`head -n 42`	Stop after the first 42 lines
`xargs -n 1 echo +`	Executes "`echo + word`" for each word in the input stream
`xargs echo 0`	Executes "`echo 0 all input stream words`" (output is now "$0 + 1 + 2 + 3 + \ldots + 42$")
`bc`	

Answer to quiz

```
yes "" | cat -n | head -n 42 | xargs -n 1 echo +
  | xargs echo 0 | bc
```

`yes ""`	Print empty lines, forever
`cat -n`	Number those empty lines
`head -n 42`	Stop after the first 42 lines
`xargs -n 1 echo +`	Executes "`echo + word`" for each word in the input stream
`xargs echo 0`	Executes "`echo 0 all input stream words`" (output is now "$0 + 1 + 2 + 3 + \ldots + 42$")
`bc`	arbitrary precision calculator

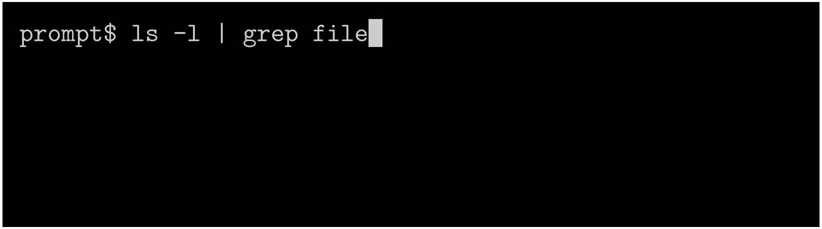
gzip: compress a file

- Usage: `gzip [file] [file]`
- Compresses each file, adds “`.gz`” to file name
- If no files specified:
  - Reads from standard input
  - Writes to standard output

```
prompt$ █
```

gzip: compress a file

- Usage: `gzip [file] [file]`
- Compresses each file, adds “`.gz`” to file name
- If no files specified:
  - Reads from standard input
  - Writes to standard output



```
prompt$ ls -l | grep file
```

gzip: compress a file

- Usage: `gzip [file] [file]`
- Compresses each file, adds “`.gz`” to file name
- If no files specified:
  - Reads from standard input
  - Writes to standard output

```
prompt$ ls -l | grep file
-rw------- 1 alice staff 62976 Jul 24 15:58 file.c
prompt$ █
```

gzip: compress a file

- Usage: `gzip [file] [file]`
- Compresses each file, adds “.gz” to file name
- If no files specified:
  - Reads from standard input
  - Writes to standard output

```
prompt$ ls -l | grep file
-rw------- 1 alice staff 62976 Jul 24 15:58 file.c
prompt$ gzip file.c
```

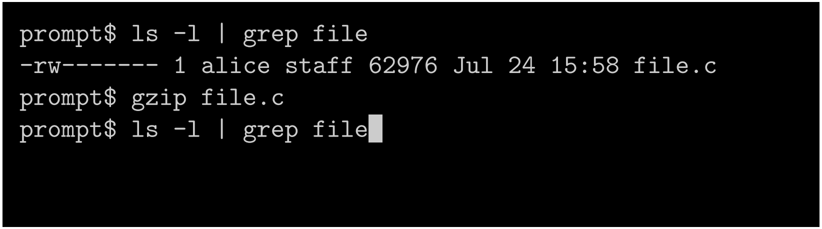
gzip: compress a file

- Usage: `gzip [file] [file]`
- Compresses each file, adds ".gz" to file name
- If no files specified:
  - Reads from standard input
  - Writes to standard output

```
prompt$ ls -l | grep file
-rw------- 1 alice staff 62976 Jul 24 15:58 file.c
prompt$ gzip file.c
prompt$ 
```

gzip: compress a file

- Usage: `gzip [file] [file]`
- Compresses each file, adds “`.gz`” to file name
- If no files specified:
  - Reads from standard input
  - Writes to standard output



```
prompt$ ls -l | grep file
-rw------- 1 alice staff 62976 Jul 24 15:58 file.c
prompt$ gzip file.c
prompt$ ls -l | grep file
```

gzip: compress a file

- Usage: `gzip [file] [file]`
- Compresses each file, adds “.gz” to file name
- If no files specified:
  - Reads from standard input
  - Writes to standard output

```
prompt$ ls -l | grep file
-rw------- 1 alice staff 62976 Jul 24 15:58 file.c
prompt$ gzip file.c
prompt$ ls -l | grep file
-rw------- 1 alice staff 14458 Jul 24 15:58 file.c.gz
prompt$ 
```

Some thoughts on compression

- Is the compressed file always smaller?

Some thoughts on compression

- Is the compressed file always smaller?
- No. Some files do not compress well

Some thoughts on compression

- Is the compressed file always smaller?
- No. Some files do not compress well
  - Small files

Some thoughts on compression

- Is the compressed file always smaller?
- No. Some files do not compress well
  - Small files
  - Files that are already compressed
    - Some file formats are compressed: `jpeg`, `mp3`, . . .
    - Files that have already been gzipped

Some thoughts on compression

- Is the compressed file always smaller?
- No. Some files do not compress well
  - Small files
  - Files that are already compressed
    - Some file formats are compressed: `jpeg`, `mp3`, . . .
    - Files that have already been gzipped
  - By default, `gzip` will not compress a file ending with “`.gz`”

Some thoughts on compression

- Is the compressed file always smaller?
- No. Some files do not compress well
  - Small files
  - Files that are already compressed
    - Some file formats are compressed: `jpeg`, `mp3`, ...
    - Files that have already been gzipped
  - By default, `gzip` will not compress a file ending with “`.gz`”
  - Let’s defeat that and see what happens

```
prompt$ 
```

Some thoughts on compression

- Is the compressed file always smaller?
- No. Some files do not compress well
  - Small files
  - Files that are already compressed
    - Some file formats are compressed: jpeg, mp3, . . .
    - Files that have already been gzipped
  - By default, gzip will not compress a file ending with “.gz”
  - Let’s defeat that and see what happens

```
prompt$ mv file.c.gz file.c.g
```

Some thoughts on compression

- Is the compressed file always smaller?
- No. Some files do not compress well
  - Small files
  - Files that are already compressed
    - Some file formats are compressed: `jpeg`, `mp3`, . . .
    - Files that have already been gzipped
  - By default, `gzip` will not compress a file ending with "`.gz`"
  - Let's defeat that and see what happens

```
prompt$ mv file.c.gz file.c.g
prompt$ █
```

Some thoughts on compression

- Is the compressed file always smaller?
- No. Some files do not compress well
  - Small files
  - Files that are already compressed
    - Some file formats are compressed: jpeg, mp3, . . .
    - Files that have already been gzipped
  - By default, gzip will not compress a file ending with “.gz”
  - Let’s defeat that and see what happens

```
prompt$ mv file.c.gz file.c.g
prompt$ gzip file.c.g
```

Some thoughts on compression

- Is the compressed file always smaller?
- No. Some files do not compress well
  - Small files
  - Files that are already compressed
    - Some file formats are compressed: `jpeg`, `mp3`, ...
    - Files that have already been gzipped
  - By default, `gzip` will not compress a file ending with "`.gz`"
  - Let's defeat that and see what happens

```
prompt$ mv file.c.gz file.c.g
prompt$ gzip file.c.g
prompt$ 
```

Some thoughts on compression

- Is the compressed file always smaller?
- No. Some files do not compress well
  - Small files
  - Files that are already compressed
    - Some file formats are compressed: `jpeg`, `mp3`, ...
    - Files that have already been gzipped
  - By default, `gzip` will not compress a file ending with "`.gz`"
  - Let's defeat that and see what happens

```
prompt$ mv file.c.gz file.c.g
prompt$ gzip file.c.g
prompt$ ls -l | grep file
```

Some thoughts on compression

- Is the compressed file always smaller?
- No. Some files do not compress well
    - Small files
    - Files that are already compressed
        - Some file formats are compressed: `jpeg`, `mp3`, ...
        - Files that have already been gzipped
    - By default, `gzip` will not compress a file ending with "`.gz`"
    - Let's defeat that and see what happens

```
prompt$ mv file.c.gz file.c.g
prompt$ gzip file.c.g
prompt$ ls -l | grep file
-rw------- 1 alice staff 14490 Jul 24 15:58 file.c.g.gz
prompt$ █
```

gunzip: uncompress a file

- Usage: `gunzip [file.gz] [file.gz]`
- Uncompresses each file, removes “.gz” from file name
- If no files specified:
  - Reads from standard input
  - Writes to standard output

```
prompt$ █
```

gunzip: uncompress a file

- Usage: `gunzip [file.gz] [file.gz]`
- Uncompresses each file, removes “`.gz`” from file name
- If no files specified:
  - Reads from standard input
  - Writes to standard output

```
prompt$ gunzip file.c.g.gz
```

gunzip: uncompress a file

- Usage: `gunzip [file.gz] [file.gz]`
- Uncompresses each file, removes “.gz” from file name
- If no files specified:
    - Reads from standard input
    - Writes to standard output

```
prompt$ gunzip file.c.g.gz
prompt$ █
```

gunzip: uncompress a file

- Usage: `gunzip [file.gz] [file.gz]`
- Uncompresses each file, removes “`.gz`” from file name
- If no files specified:
  - Reads from standard input
  - Writes to standard output

```
prompt$ gunzip file.c.g.gz
prompt$ mv file.c.g file.c.gz
```

gunzip: uncompress a file

- Usage: `gunzip [file.gz] [file.gz]`
- Uncompresses each file, removes "`.gz`" from file name
- If no files specified:
  - Reads from standard input
  - Writes to standard output

```
prompt$ gunzip file.c.g.gz
prompt$ mv file.c.g file.c.gz
prompt$ █
```

gunzip: uncompress a file

- Usage: `gunzip [file.gz] [file.gz]`
- Uncompresses each file, removes “`.gz`” from file name
- If no files specified:
  - Reads from standard input
  - Writes to standard output

```
prompt$ gunzip file.c.g.gz
prompt$ mv file.c.g file.c.gz
prompt$ gunzip file.c.gz
```

gunzip: uncompress a file

- Usage: `gunzip [file.gz] [file.gz]`
- Uncompresses each file, removes “`.gz`” from file name
- If no files specified:
  - Reads from standard input
  - Writes to standard output

```
prompt$ gunzip file.c.g.gz
prompt$ mv file.c.g file.c.gz
prompt$ gunzip file.c.gz
prompt$ █
```

gunzip: uncompress a file

- Usage: `gunzip [file.gz] [file.gz]`
- Uncompresses each file, removes "`.gz`" from file name
- If no files specified:
  - Reads from standard input
  - Writes to standard output

```
prompt$ gunzip file.c.g.gz
prompt$ mv file.c.g file.c.gz
prompt$ gunzip file.c.gz
prompt$ ls -l | grep file
```

gunzip: uncompress a file

- Usage: `gunzip [file.gz] [file.gz]`
- Uncompresses each file, removes “.gz” from file name
- If no files specified:
  - Reads from standard input
  - Writes to standard output

```
prompt$ gunzip file.c.g.gz
prompt$ mv file.c.g file.c.gz
prompt$ gunzip file.c.gz
prompt$ ls -l | grep file
-rw------- 1 alice staff 62976 Jul 24 15:58 file.c
prompt$
```

zcat: show a compressed file

- Usage: `zcat [file.gz] [file.gz]`
- Uncompresses, then "`cats`", the specified files
- Files are not modified
- If no files specified: reads from standard input

```
prompt$ █
```

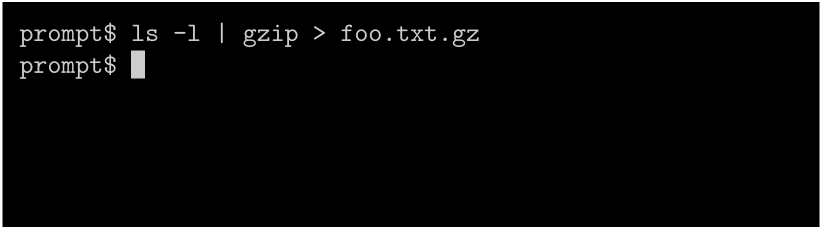
zcat: show a compressed file

- Usage: `zcat [file.gz] [file.gz]`
- Uncompresses, then "cats", the specified files
- Files are not modified
- If no files specified: reads from standard input

```
prompt$ ls -l | gzip > foo.txt.gz
```

zcat: show a compressed file

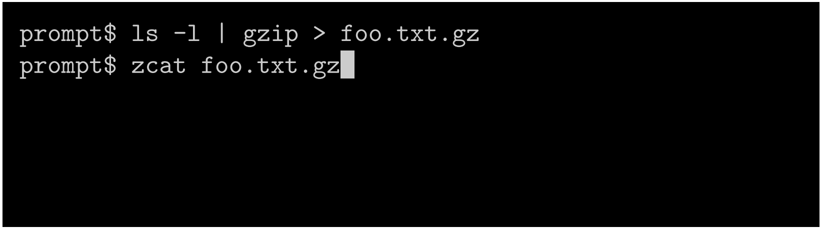
- Usage: `zcat [file.gz] [file.gz]`
- Uncompresses, then "cats", the specified files
- Files are not modified
- If no files specified: reads from standard input



```
prompt$ ls -l | gzip > foo.txt.gz
prompt$ █
```

zcat: show a compressed file

- Usage: `zcat [file.gz] [file.gz]`
- Uncompresses, then "cats", the specified files
- Files are not modified
- If no files specified: reads from standard input



```
prompt$ ls -l | gzip > foo.txt.gz
prompt$ zcat foo.txt.gz
```

zcat: show a compressed file

- Usage: `zcat [file.gz] [file.gz]`
- Uncompresses, then "cats", the specified files
- Files are not modified
- If no files specified: reads from standard input

```
prompt$ ls -l | gzip > foo.txt.gz
prompt$ zcat foo.txt.gz
total 64
-rw------- 1 alice staff 62976 Jul 24 15:58 file.c
-rw------- 1 alice staff     0 Sep 18 13:42 foo.txt.gz
prompt$ 
```

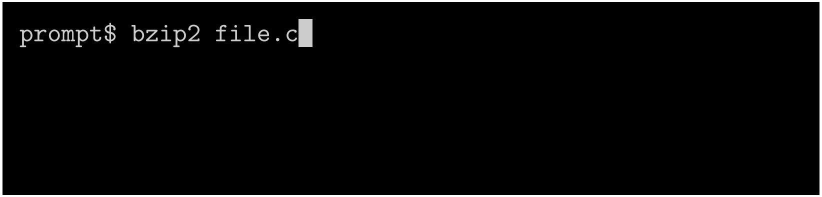
bzip2, bunzip2, bzcat

- Utilities are similar to gzip, gunzip, gzcat
- Extension used is “.bz2”
- Use a different compression algorithm
- Tends to be more compact, especially for large files

```
prompt$ █
```

bzip2, bunzip2, bzcat

- Utilities are similar to gzip, gunzip, gzcat
- Extension used is “.bz2’
- Use a different compression algorithm
- Tends to be more compact, especially for large files



```
prompt$ bzip2 file.c
```

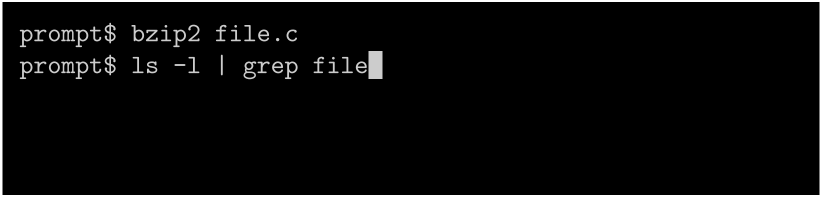
bzip2, bunzip2, bzcat

- Utilities are similar to gzip, gunzip, gzcat
- Extension used is “.bz2”
- Use a different compression algorithm
- Tends to be more compact, especially for large files

```
prompt$ bzip2 file.c
prompt$ █
```

bzip2, bunzip2, bzcat

- Utilities are similar to gzip, gunzip, gzcat
- Extension used is ".bz2'
- Use a different compression algorithm
- Tends to be more compact, especially for large files



```
prompt$ bzip2 file.c
prompt$ ls -l | grep file
```

bzip2, bunzip2, bzcat

- Utilities are similar to gzip, gunzip, gzcat
- Extension used is “.bz2”
- Use a different compression algorithm
- Tends to be more compact, especially for large files

```
prompt$ bzip2 file.c
prompt$ ls -l | grep file
-rw------- 1 alice staff 13519 Jul 24 15:58 file.c.bz2
prompt$
```

bzip2, bunzip2, bzcat

- Utilities are similar to gzip, gunzip, gzcat
- Extension used is ".bz2'
- Use a different compression algorithm
- Tends to be more compact, especially for large files

```
prompt$ bzip2 file.c
prompt$ ls -l | grep file
-rw------- 1 alice staff 13519 Jul 24 15:58 file.c.bz2
prompt$ bunzip2 file.c.bz2
```

bzip2, bunzip2, bzcat

- Utilities are similar to gzip, gunzip, gzcat
- Extension used is ".bz2"
- Use a different compression algorithm
- Tends to be more compact, especially for large files

```
prompt$ bzip2 file.c
prompt$ ls -l | grep file
-rw------- 1 alice staff 13519 Jul 24 15:58 file.c.bz2
prompt$ bunzip2 file.c.bz2
prompt$ 
```

Misc. utilities

`cat -n`	show files and number lines
`grep`	select lines matching text
`head`	select beginning of a file
`less`	modern pager
`more`	classic pager
`sort`	sort a file
`tail`	select end of a file
`tee`	pipeline splitter
`tr`	translate characters
`wc`	count lines, words
`xargs`	extract arguments
`yes`	print “y” or other text, forever

Compression tools

`bzip2`	:	produce “`.bz2`” files
`gzip`	:	produce “`.gz`” files

Uncompression tools

`bunzip2`	:	uncompress “`.bz2`” files
`bzcat`	:	cat “`.bz2`” files
`gunzip`	:	uncompress “`.gz`” files
`zcat`	:	cat “`.gz`” files

End of lecture