Shell Programming II

15-123

Systems Skills in C and Unix

Regular Expressions

- Shell scripts can include utilities such as
  - grep
    - Pattern matching
  - sed
    - Stream editor
  - awk
    - Pattern scanning and processing

sed revisited

- A stream editor
- Offspring of the unix “ed”
- Very useful tool
  - cat file.txt | sed ‘s/<.*>//’ > file2.txt

- Syntax: # # #
  - sed ‘s <delimiter> regex <delimiter> replacement <delimiter> flags’
  - Flags: -l for local (first match) or g (global) – all matches

Inter Process Communication (IPC)

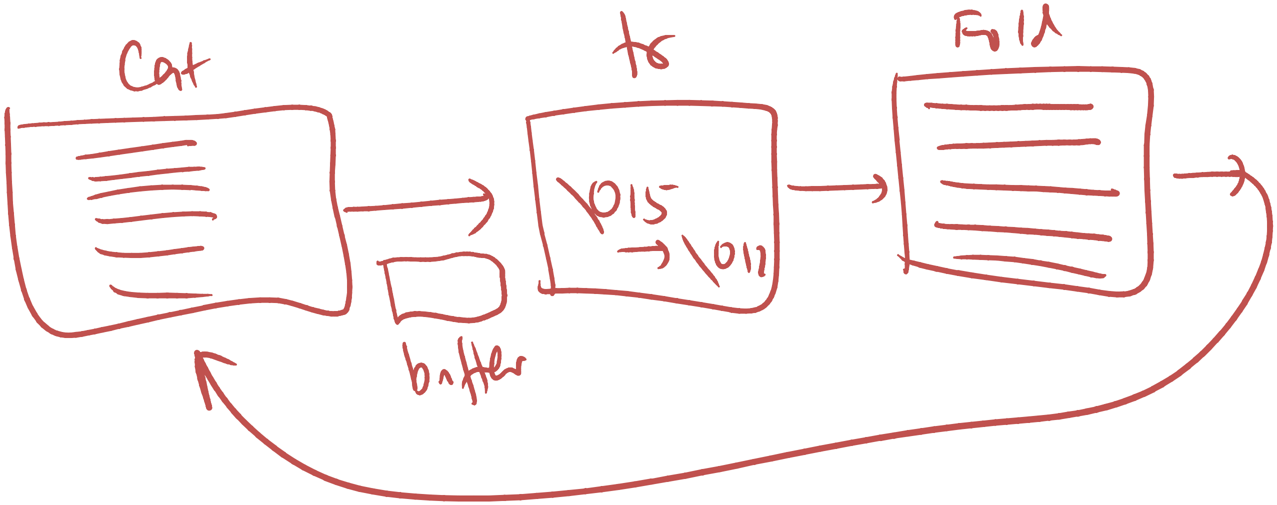
- **Pipes**
  - Creates the IPC
  - ls | sort | echo
    - 4 processes in play
- Each call spans a new process
  - Using folk

Editing in Place

- cat somefile.txt | tr -d "\015" "\012" | fold > somefile.txt

<cr> <tab>

- What does it do?
- What are some of the problems?



How does pipes work

- A finite buffer to allow communication
  - Typically 8K
- If input file is less than the buffer
  - We may be ok
- What if input file is more than the buffer
  - Redirecting output to the same file is a bad idea

How to deal with this?

- **Use a temp file**
  - **cat ${1} | tr -d "\015" "\012" | fold > ${1}.tmp mv ${1}.tmp ${1}**

    rm $1.tmp
- **Better process**
  - cat "${1}" | tr -d "\015" "\012" | fold > "/usr/tmp/${1}.$$" mv "/usr/tmp/${1}.$$" "${1}“
- **/usr/tmp** is cleared upon reboot

Pipes, Loops and Sub shells

```
#!/bin/sh
FILE=${1}
cat ${FILE}  |
while read value
 do
   echo ${value}
  done
```

Shell

Subshell

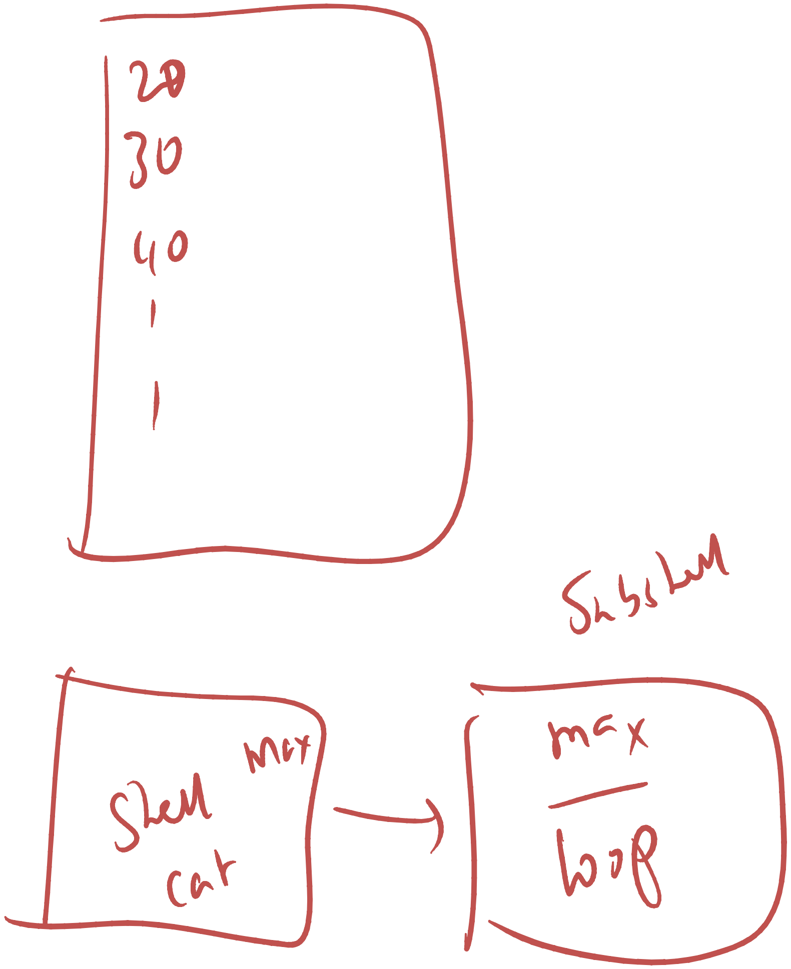
- while loop in a sub shell

What is the problem?

```
#!/bin/sh
FILE=${1}
max=0
cat ${FILE} |
 while read value
  do
      if [ ${value} -gt ${max} ];
         then
             max=${value}
             echo $max
      fi
  done
echo ${max}
```

Shell

Subshell



The fix

```
#!/bin/sh
FILE=${1}
max=0
values=`cat ${FILE}`
for value in ${values}
do    if [ ${value} -gt ${max} ];
   then
         max=${value}
    fi
    done
echo ${max}
```

Arrays in bash

```
array[2]=23
array[3]=45
array[1]=4
```

To dereference an array variable, we can use, for example

```
echo  ${array[1]}
```

Array elements need not be consecutive and some members of the array can be left uninitialized. Here is an example of printing an array in bash. Note the C style loop. Also note the spaces between tokens.

```
for ((  i=1  ;  i<=3  ;  i++  ))
do
  echo ${array[$i]}
done
```

Coding Examples