

CS395T: Introduction to Scientific and Technical Computing

Instructors:

Dr. Karl W. Schulz, Research Associate, TACC

Dr. Victor Eijkhout, Research Scientist, TACC



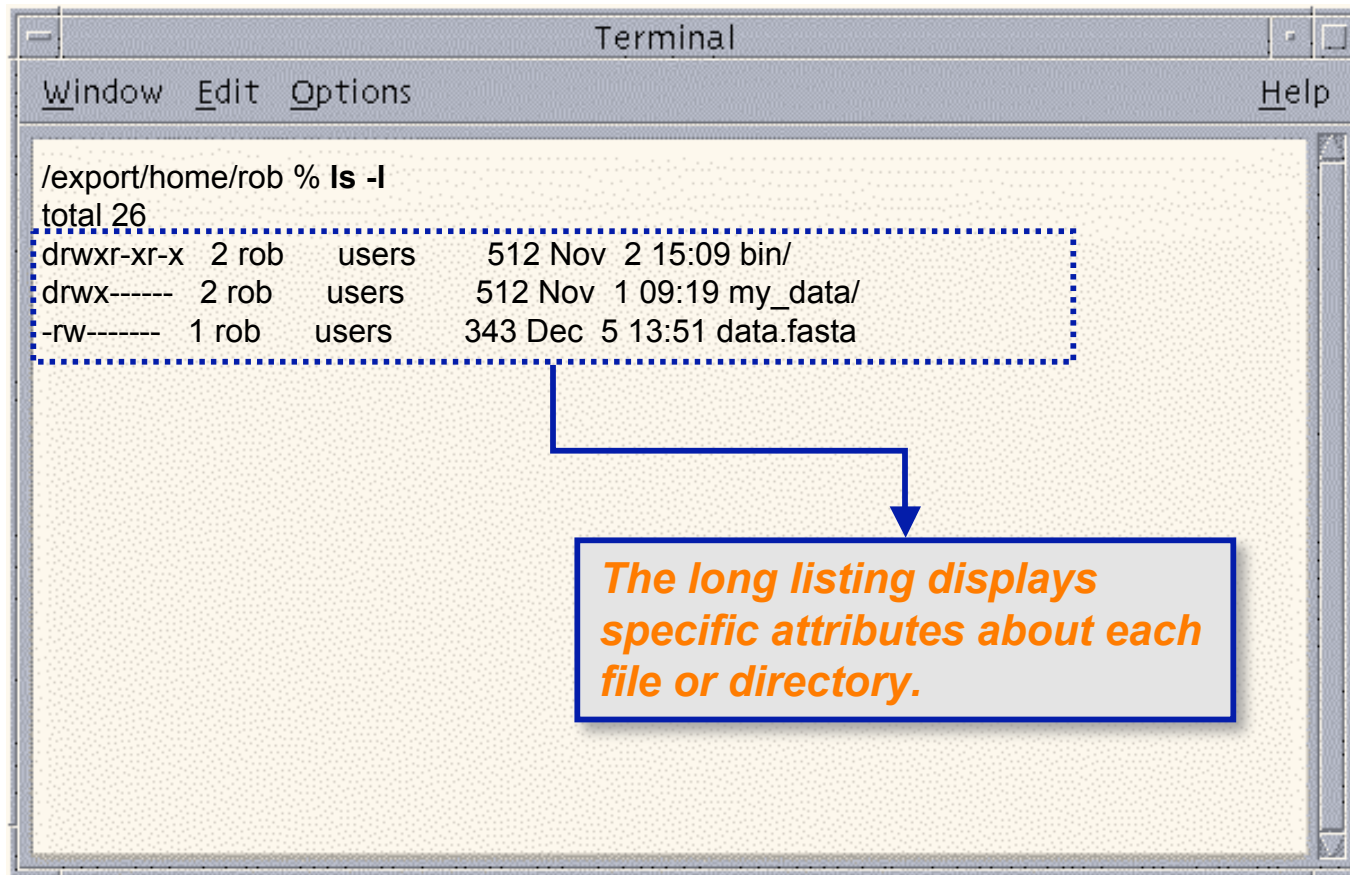
THE UNIVERSITY OF TEXAS AT AUSTIN
Texas Advanced Computing Center

Outline

- Continue with Unix overview
 - File attributes and permissions
 - Basic commands
 - Pattern matching, regular expressions
 - Shell scripting

UNIX Command Examples

Remember the “**ls -l**” command to show long listings?



```
Terminal
Window Edit Options Help

/export/home/rob % ls -l
total 26
drwxr-xr-x  2 rob  users  512 Nov  2 15:09 bin/
drwx----- 2 rob  users  512 Nov  1 09:19 my_data/
-rw----- 1 rob  users  343 Dec  5 13:51 data.fasta
```

The long listing displays specific attributes about each file or directory.

File Attributes

- Every file has a specific list of attributes:
 - Access Times:
 - when the file was created
 - when the file was last changed
 - when the file was last read
 - Size
 - Owners
 - user (*remember UID*)
 - group (*remember GID*)
 - Permissions

File Time Attributes

- Time Attributes:
 - `ls -l` shows when the file was last changed
 - `ls -lc` shows when the file was created
 - `ls -lu` shows when the file was last accessed
- Special names exist for these date-related attributes:
 - `mtime` (last modification time)
 - `ctime` (last change time, ie. when changes were made to the file or directory's inode: owner, permissions, etc.)
 - `atime` (last access time)
 - Display with 'stat' command

File Permissions

- Each file has a set of permissions that control who can *access* the file
- There are three different types of permissions:
 - read abbreviated *r*
 - write abbreviated *w*
 - execute abbreviated *x*
- In Unix, there are permission levels associated with three types of people that might access a file:
 - owner (you)
 - group (a group of other users that you set up)
 - world (anyone else browsing around on the file system)

File Permissions Display Format

— **rwxrwxrwx**

Owner Group Others

The first entry specifies the type of file:
“-” is a plain file
“d” is a directory
“c” is a character device
“b” is a block device
“l” is a symbolic link

What is this *rwX* Crazyiness?

- Meaning for Files:
 - r* – allowed to read
 - w* – allowed to write
 - x* – allowed to execute
- Meaning for Directories:
 - r* – allowed to see the names of the files
 - w* – allowed to add and remove files
 - x* – allowed to enter the directory

Changing File Permissions

- The `chmod` command changes the permissions associated with a file or directory
- Basic syntax is: `chmod mode file`
- The *mode* can be specified in two ways:
 - symbolic representation
 - octal number
- Both methods achieve the same result (*user's choice*)
- Multiple symbolic operations can be given, separated by commas

chmod: Symbolic Representation

- *Symbolic* Mode representation has the following form:

[ugoa] [+ -=] [rwxX...]

u=user
g=group
o=other
a = all

+ add permission
- remove permission
= set permission

r=read
w=write
x=execute
X= *see below*

- The **X** permission option is very handy - it sets to execute only if the file is a directory or already has execute permission

chmod Symbolic Mode Examples

```
> ls -al foo
```

```
-rw----- 1 karl support ...
```

```
> chmod g=rw foo
```

```
> ls -al foo
```

```
-rw-rw---- 1 karl support ...
```

```
> chmod u-w,g+x,o=x foo
```

```
> ls -al foo
```

```
-r--rwx--x 1 karl support ...
```

chmod: Octal Representation

- Octal Mode uses a single argument string which describes the permissions for a file (3 digits)
- Each digit of this number is a code for each of the three permission levels (user,group,world)
- Permissions are set according to the following numbers:
 - Read = 4
 - Write = 2
 - Execute = 1
- Sum the individual permissions to get the desired combination

*0 = no permissions whatsoever;
1 = execute only
2 = write only
3 = write and execute (1+2)
4 = read only
5 = read and execute (4+1)
6 = read and write (4+2)
7 = read and write and execute (4+2+1)*

chmod Octal Mode Examples

```
> ls -al foo
```

```
-rw----- 1 karl support ...
```

```
> chmod 660 foo
```

```
> ls -al foo
```

```
-rw-rw---- 1 karl support ...
```

```
> chmod 417 foo
```

```
> ls -al foo
```

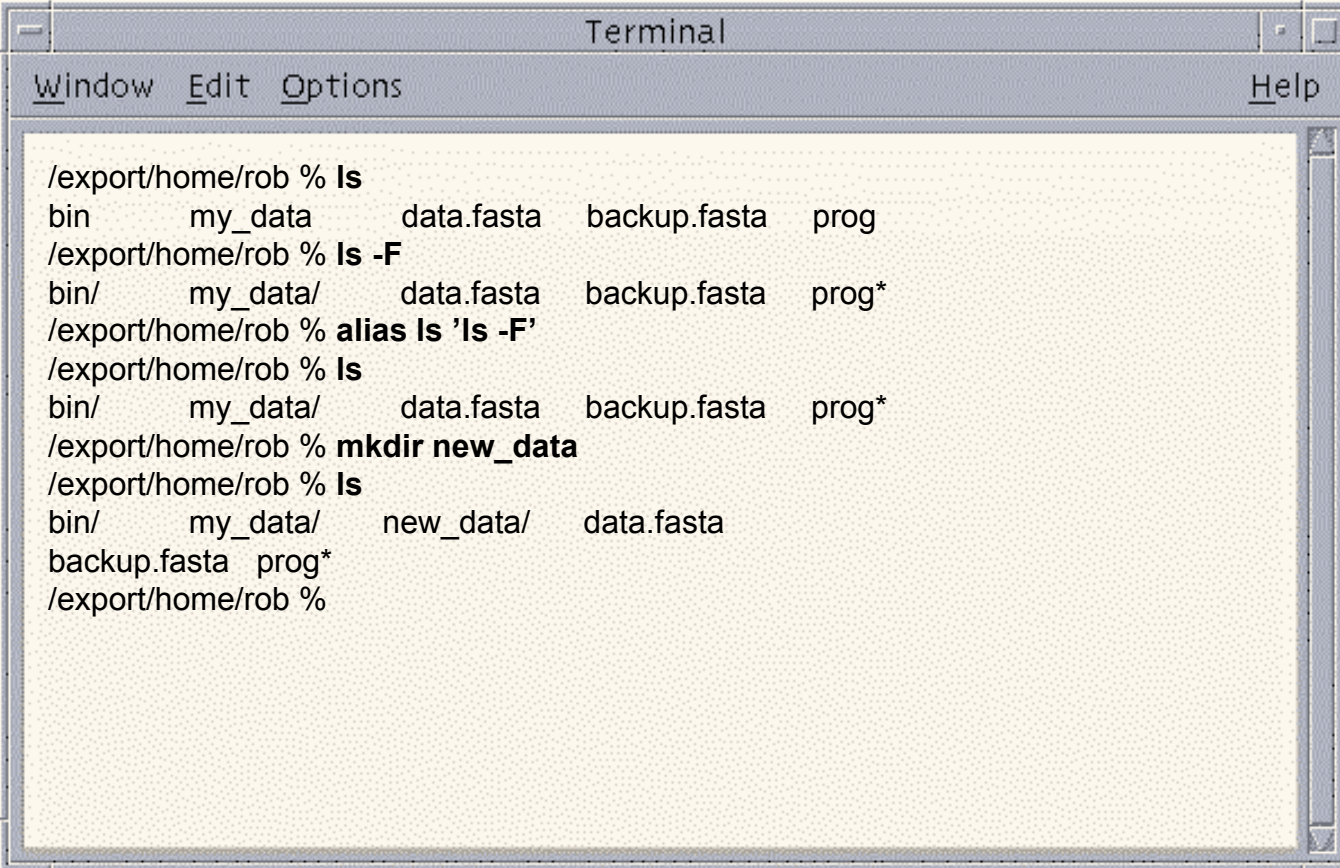
```
-r-----xrw 1 karl support ...
```

Basic Commands

- Some basic commands for interacting with the Unix file system are:
 - ls
 - cd
 - df
 - cat
 - more (less)
 - head
 - pwd
 - cp
 - awk
 - rm
 - chmod
 - tail
- touch
 - mkdir
 - rmdir
 - find
 - grep
 - chown/chgrp
- Let's cruise through some examples....

UNIX Commands: **mkdir**

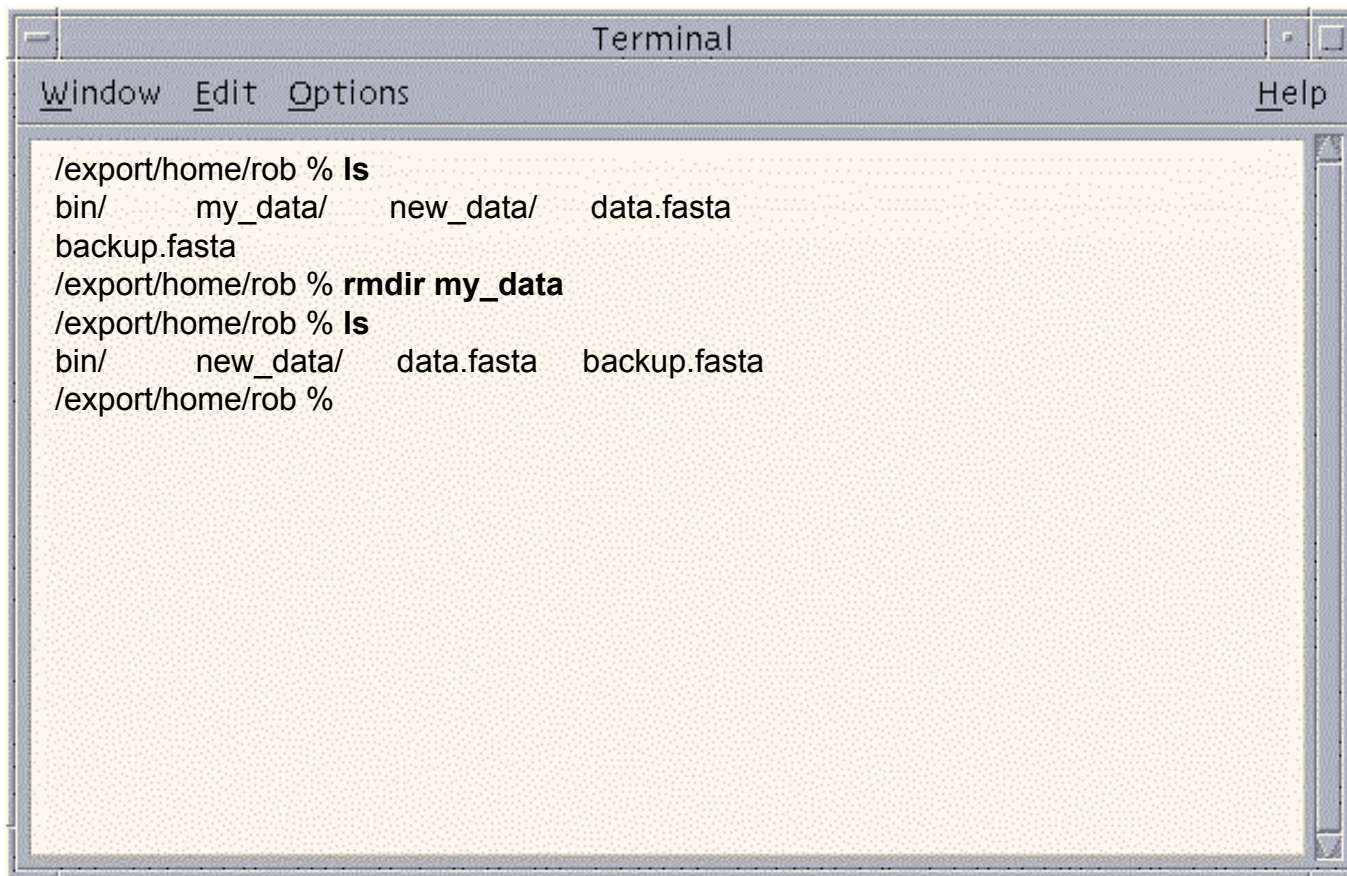
mkdir creates directories.

A terminal window titled "Terminal" with a menu bar containing "Window", "Edit", "Options", and "Help". The terminal shows a series of commands and their outputs. The user is in the directory /export/home/rob. They run 'ls' and see a list of files and directories. Then they run 'ls -F' to see file types. Next, they run 'alias ls 'ls -F'' to make 'ls' always show file types. Then they run 'mkdir new_data' to create a new directory. Finally, they run 'ls' again to see the new directory in the list.

```
/export/home/rob % ls
bin      my_data  data.fasta  backup.fasta  prog
/export/home/rob % ls -F
bin/     my_data/  data.fasta  backup.fasta  prog*
/export/home/rob % alias ls 'ls -F'
/export/home/rob % ls
bin/     my_data/  data.fasta  backup.fasta  prog*
/export/home/rob % mkdir new_data
/export/home/rob % ls
bin/     my_data/  new_data/   data.fasta
backup.fasta  prog*
/export/home/rob %
```

UNIX Commands: **rm**dir

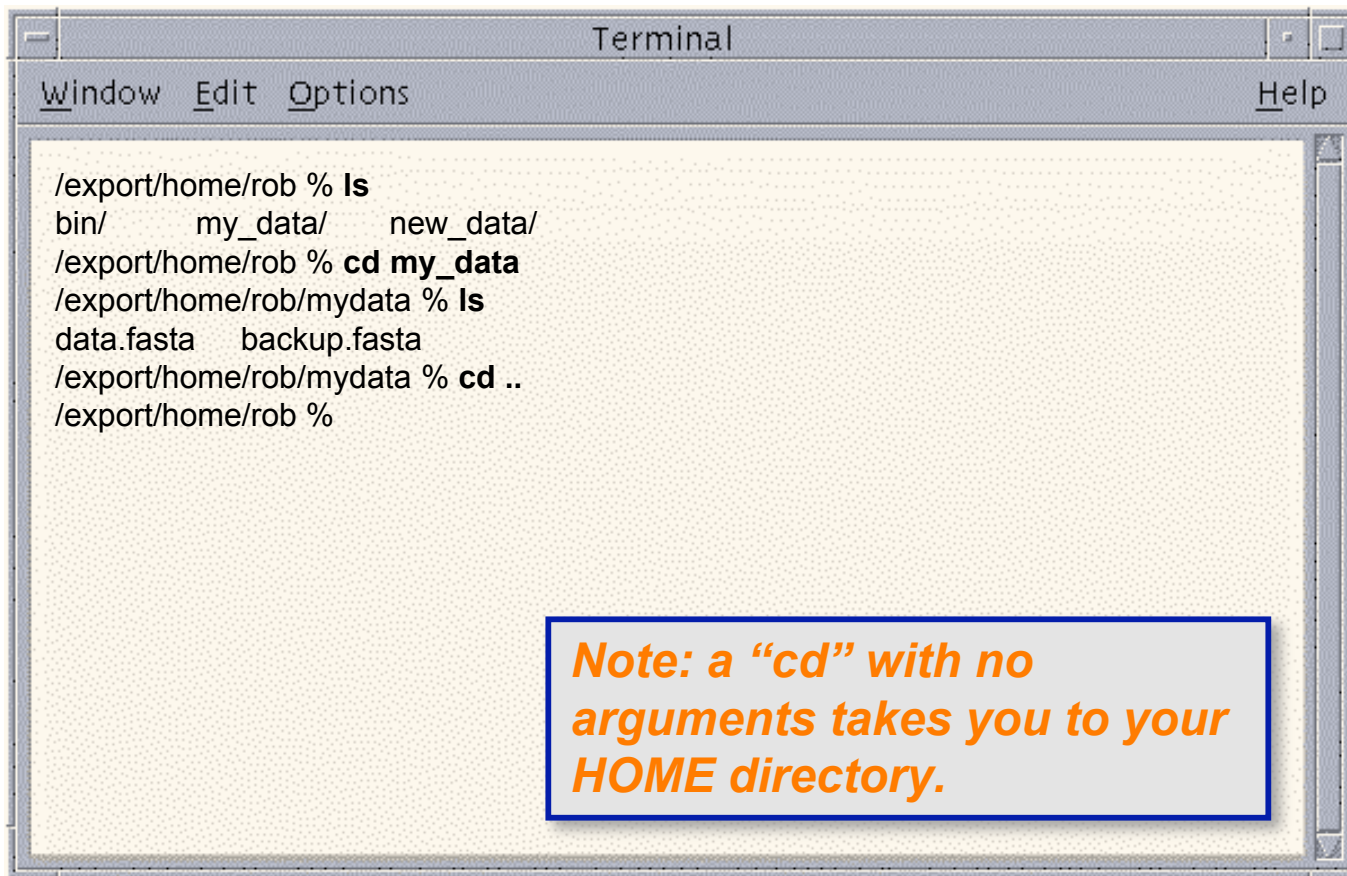
rmdir removes directories.

A screenshot of a terminal window titled "Terminal". The window has a menu bar with "Window", "Edit", "Options", and "Help". The terminal output shows a user at the prompt "/export/home/rob %". They enter "ls", and the output is "bin/ my_data/ new_data/ data.fasta backup.fasta". Then they enter "rm -r my_data". Finally, they enter "ls" again, and the output is "bin/ new_data/ data.fasta backup.fasta". The prompt is now "/export/home/rob %".

```
/export/home/rob % ls
bin/      my_data/   new_data/  data.fasta
backup.fasta
/export/home/rob % rm -r my_data
/export/home/rob % ls
bin/      new_data/  data.fasta backup.fasta
/export/home/rob %
```


UNIX Commands: **cd**

cd changes the current directory.



A terminal window titled "Terminal" with a menu bar containing "Window", "Edit", "Options", and "Help". The terminal shows a series of commands and their outputs:

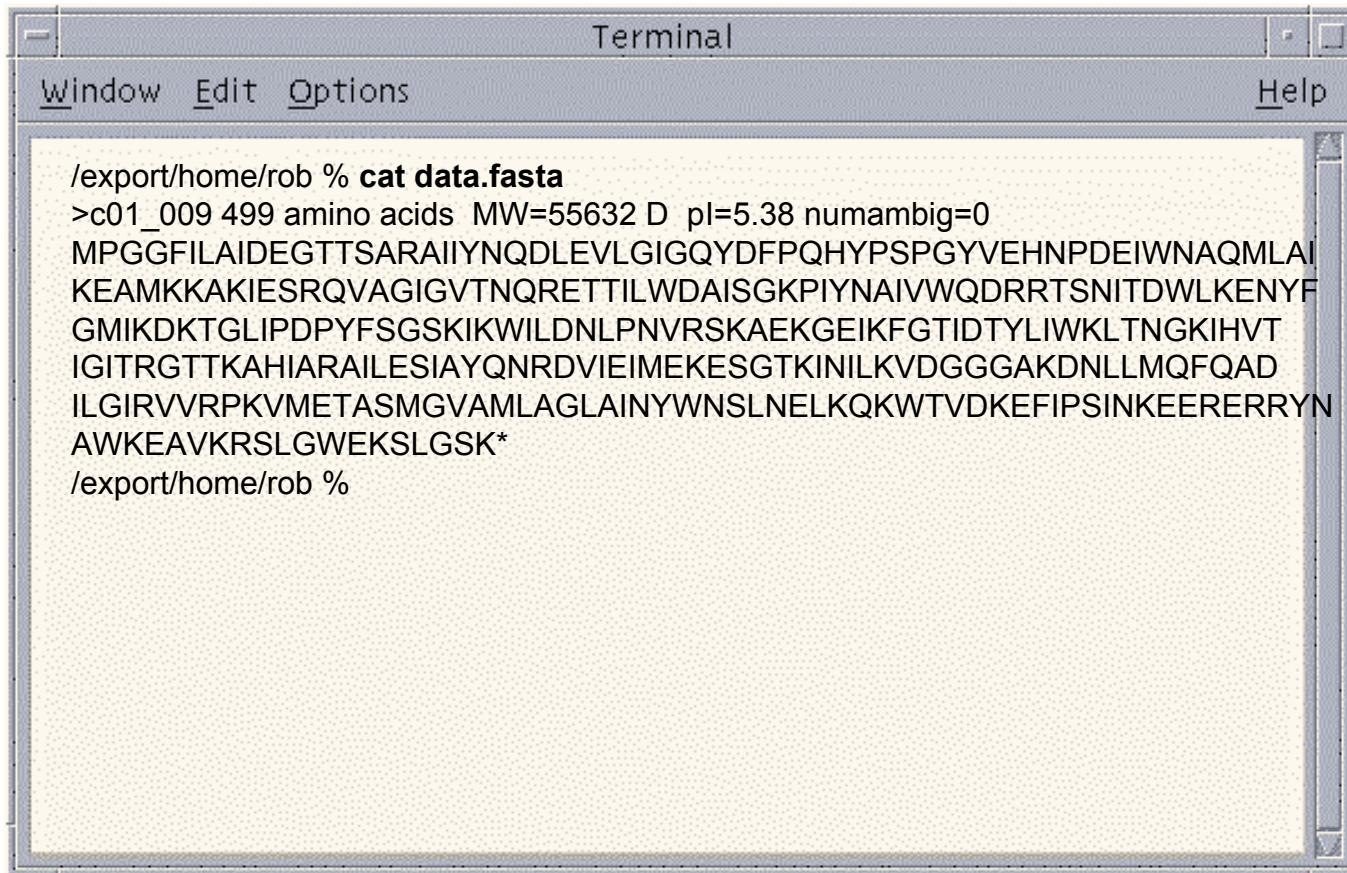
```
/export/home/rob % ls  
bin/      my_data/    new_data/  
/export/home/rob % cd my_data  
/export/home/rob/mydata % ls  
data.fasta  backup.fasta  
/export/home/rob/mydata % cd ..  
/export/home/rob %
```

In the bottom right corner of the terminal window, there is a blue-bordered box containing the following text:

Note: a "cd" with no arguments takes you to your HOME directory.

UNIX Commands: **cat**

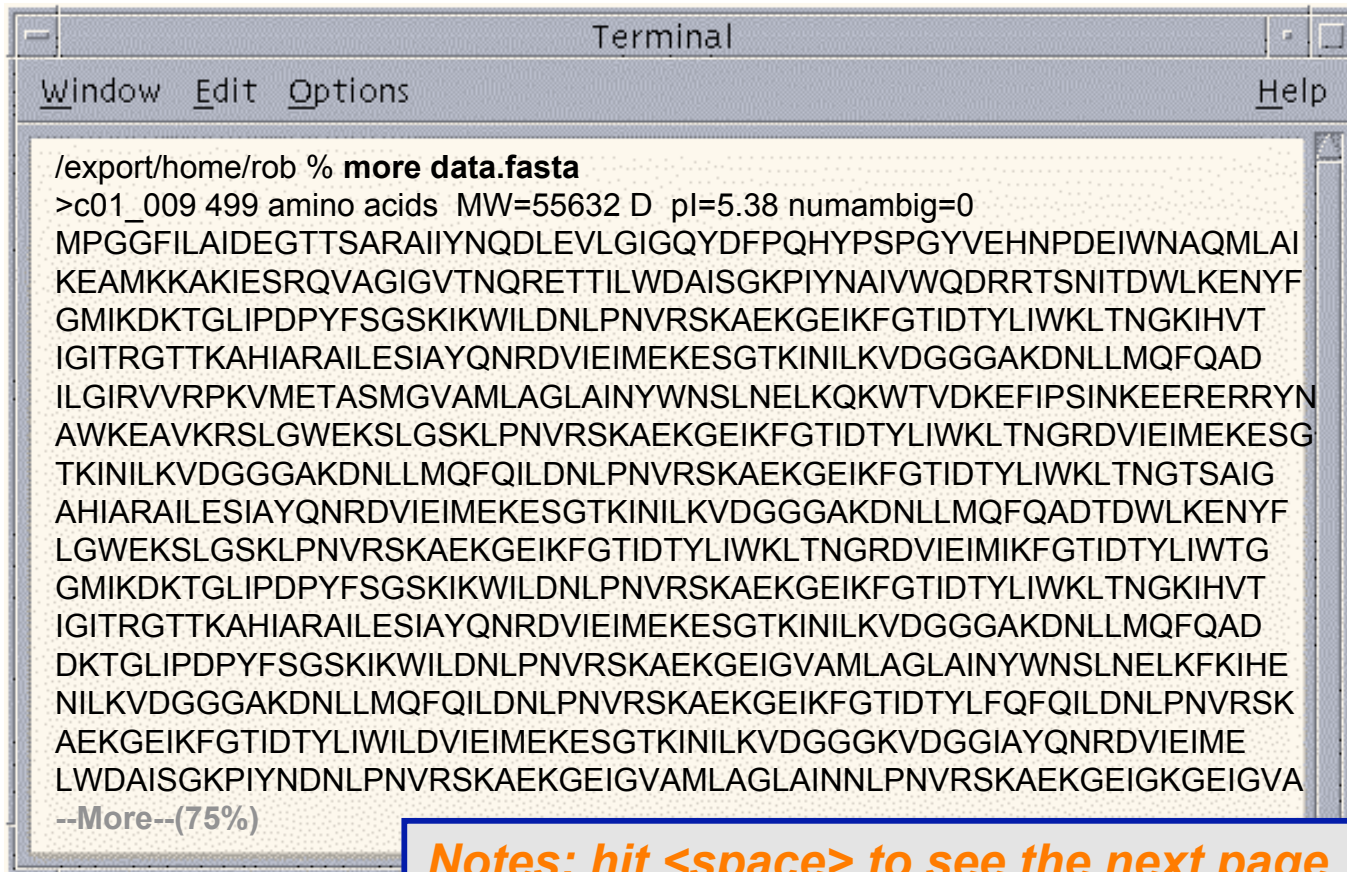
cat displays the contents of a text file:

A screenshot of a terminal window titled "Terminal". The window has a menu bar with "Window", "Edit", "Options", and "Help". The terminal text shows a user at the prompt "/export/home/rob %" typing the command "cat data.fasta". The output is a multi-line FASTA format entry for a protein. The first line is a header starting with ">c01_009" followed by protein statistics. The subsequent lines are the amino acid sequence in all caps. The sequence ends with an asterisk on a new line. The prompt "/export/home/rob %" appears again at the bottom.

```
/export/home/rob % cat data.fasta
>c01_009 499 amino acids MW=55632 D pl=5.38 numambig=0
MPGGFILAIDEGTTSARAIYNQDLEVLGIGQYDFPQHYPSPGYVEHNPDEIWNAQMLAI
KEAMKKAKIESRQVAGIGVTNQRETTILWDAISGKPIYNAIVWQDRRTSNITDWLKENYF
GMIKDKTGLIPDPYFSGSKIKWILDNLPNVRSKAEKGEIKFGTIDTYLIWKL TNGKIHVT
IGITRGTTKAHIAIRAILESIA YQNRDVIEIMEKESGTKINILKVDGGGAKDNLLMQFQAD
ILGIRVVRPKVMETASMGVAMLAGLAINYWNSL NELKQKWTVDKEFIPSINKEERERRYN
AWKEAVKRSLGWEKSLGSK*
/export/home/rob %
```

UNIX Commands: **more**

more displays the contents of a text file one screen's worth at a time:

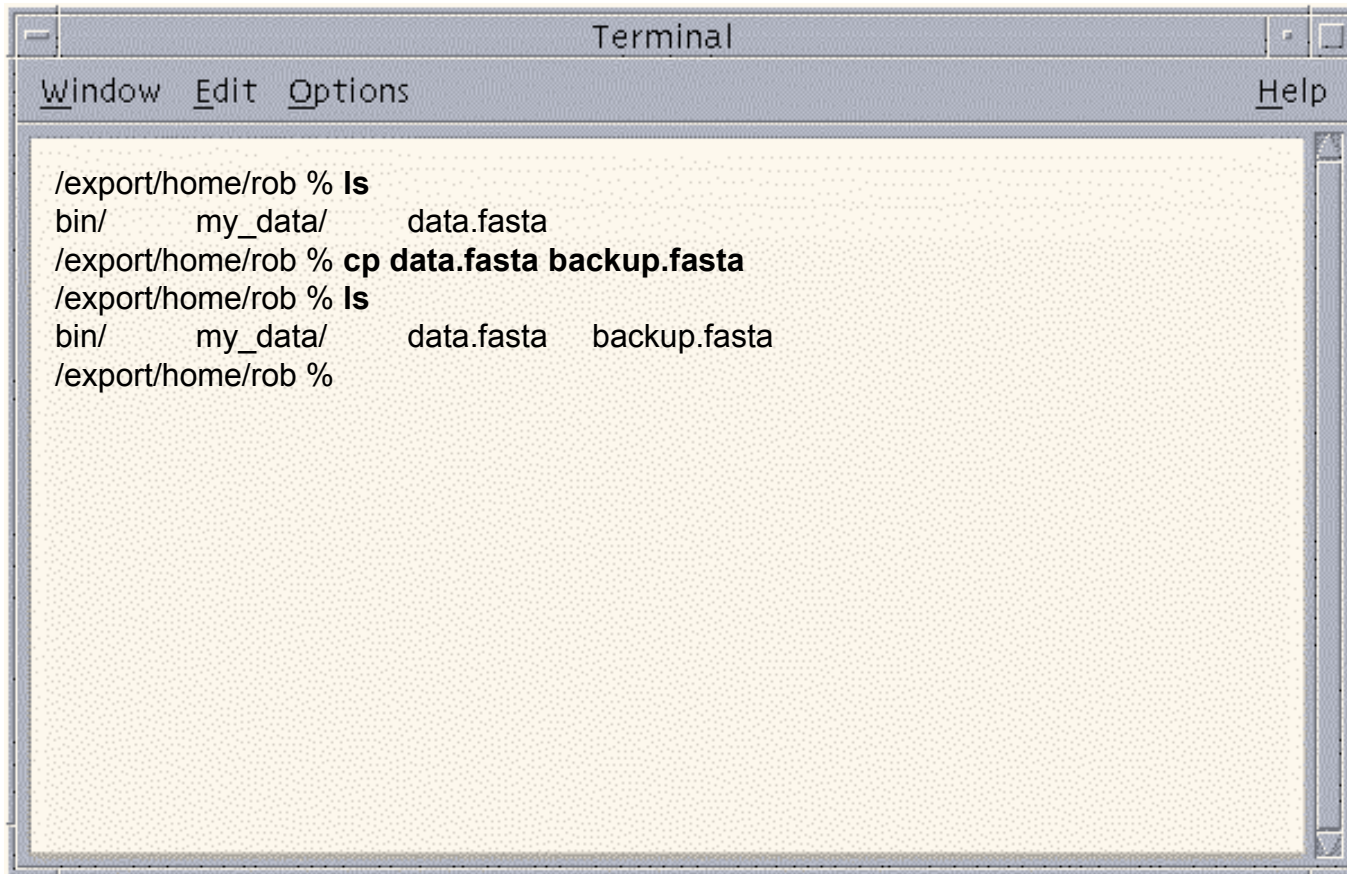
A screenshot of a terminal window titled "Terminal". The window has a menu bar with "Window", "Edit", "Options", and "Help". The terminal shows the command "/export/home/rob % **more data.fasta**" being executed. The output is a long sequence of amino acid single-letter codes, with line numbers 1 through 20 visible on the left margin. The text ends with "--More--(75%)".

```
/export/home/rob % more data.fasta
>c01_009 499 amino acids  MW=55632 D  pl=5.38 numambig=0
1  MPGGFILAIDEGTTSARAIYNQDLEVLGIGQYDFPQHYPSPGYVEHNPDEIWNAQMLAI
2  KEAMKKAKIESRQVAGIGVTNQRETTILWDAISGKPIYNAIVWQDRRTSNITDWLKENYF
3  GMIKDKTGLIPDPYFSGSKIKWILDNLPNVRSKAEKGEIKFGTIDTYLIWKL TNGKIHVT
4  IGITRGTTKAHIARAILESIAYQNRDVIEIMEKESGTKINILKVDGGGAKDNLLMQFQAD
5  ILGIRVVRPKVMETASMGVAMLAGLAINYWNSLNEKQKWTVDKEFIPSINKEERERRYN
6  AWKEAVKRSLGWEKSLGSKLPNVRSKAEKGEIKFGTIDTYLIWKL TNGRDVIEIMEKESG
7  TKINILKVDGGGAKDNLLMQFQILDNLPNVRSKAEKGEIKFGTIDTYLIWKL TNGTSAIG
8  AHIARAILESIAYQNRDVIEIMEKESGTKINILKVDGGGAKDNLLMQFQADTDWLKENYF
9  LGWEKSLGSKLPNVRSKAEKGEIKFGTIDTYLIWKL TNGRDVIEIMIKFGTIDTYLIWTG
10 GMIKDKTGLIPDPYFSGSKIKWILDNLPNVRSKAEKGEIKFGTIDTYLIWKL TNGKIHVT
11 IGITRGTTKAHIARAILESIAYQNRDVIEIMEKESGTKINILKVDGGGAKDNLLMQFQAD
12 DKTGLIPDPYFSGSKIKWILDNLPNVRSKAEKGEIGVAMLAGLAINYWNSLNEKFKIHE
13 NILKVDGGGAKDNLLMQFQILDNLPNVRSKAEKGEIKFGTIDTYLFQFQILDNLPNVRSK
14 AEKGEIKFGTIDTYLIWILDVIEIMEKESGTKINILKVDGGGKVDGGIAYQNRDVIEIME
15 LWDAISGKPIYNDNLPNVRSKAEKGEIGVAMLAGLAINNLPNVRSKAEKGEIGKGEIGVA
--More--(75%)
```

Notes: hit <space> to see the next page
hit "q" to quit, "/" to search, read the man page.
"less" is an enhanced version of "more" on Linux

UNIX Commands: **cp**

cp copies files

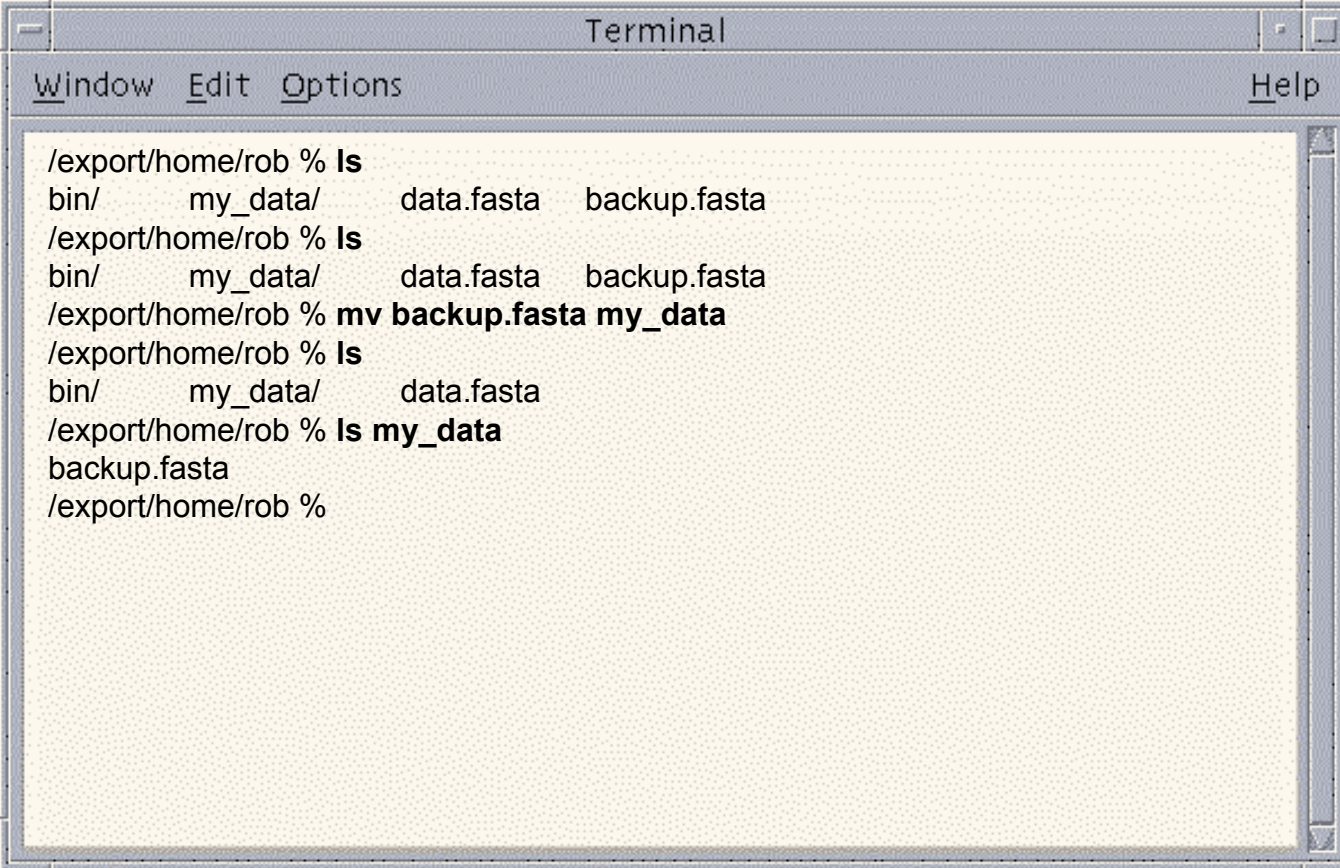
A screenshot of a terminal window titled "Terminal". The window has a menu bar with "Window", "Edit", "Options", and "Help". The terminal content shows a user at the prompt "/export/home/rob %". They first run "ls", which shows "bin/ my_data/ data.fasta". Then they run "cp data.fasta backup.fasta". Finally, they run "ls" again, which shows "bin/ my_data/ data.fasta backup.fasta".

```
Terminal
Window Edit Options Help

/export/home/rob % ls
bin/      my_data/      data.fasta
/export/home/rob % cp data.fasta backup.fasta
/export/home/rob % ls
bin/      my_data/      data.fasta  backup.fasta
/export/home/rob %
```


UNIX Commands: **mv**

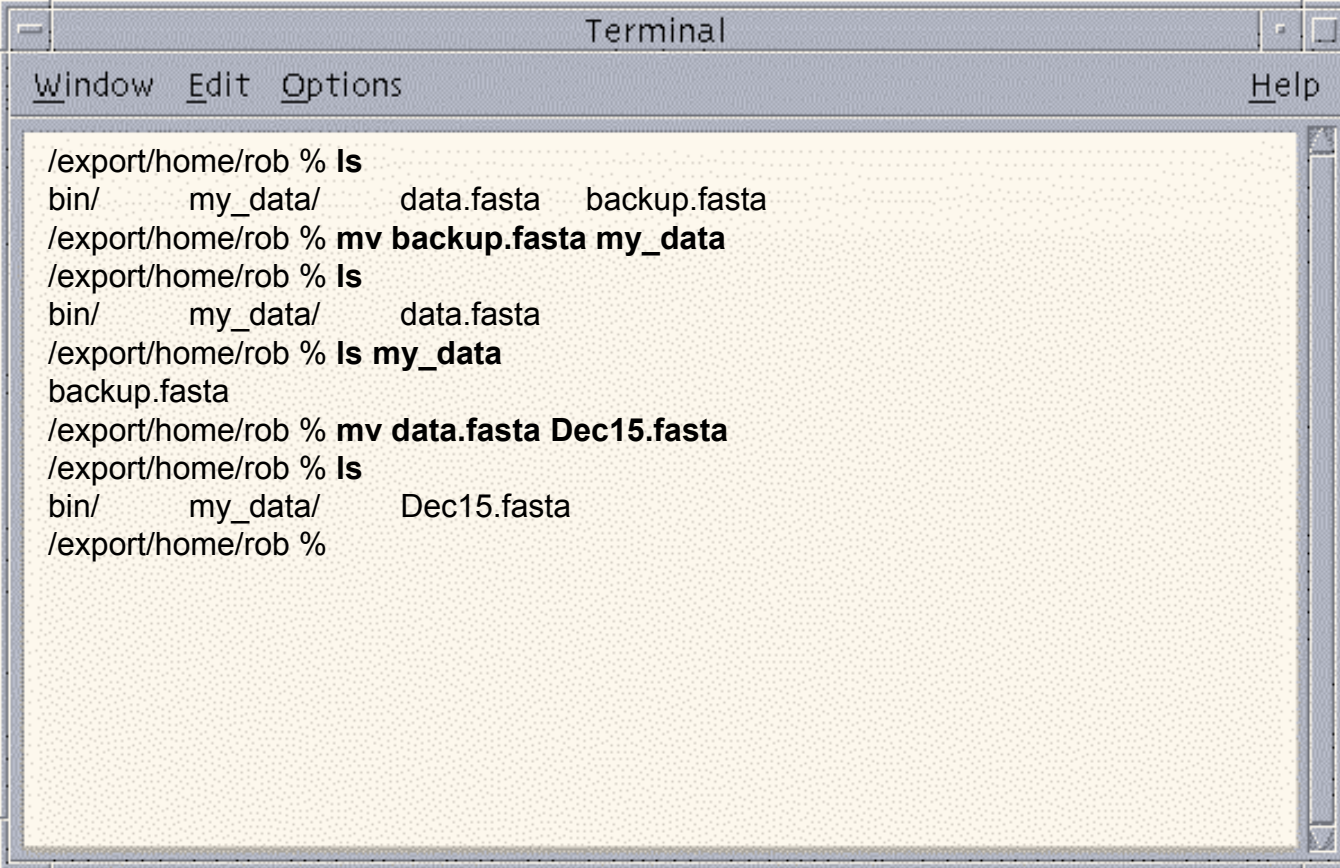
mv moves files

A screenshot of a UNIX terminal window titled "Terminal". The window has a menu bar with "Window", "Edit", "Options", and "Help". The terminal content shows a series of commands and their outputs. First, the user runs "ls" in the directory "/export/home/rob", which lists "bin/", "my_data/", "data.fasta", and "backup.fasta". Then, the user runs "mv backup.fasta my_data", which moves the file. Finally, the user runs "ls my_data", which lists "backup.fasta".

```
/export/home/rob % ls
bin/      my_data/  data.fasta  backup.fasta
/export/home/rob % ls
bin/      my_data/  data.fasta  backup.fasta
/export/home/rob % mv backup.fasta my_data
/export/home/rob % ls
bin/      my_data/  data.fasta
/export/home/rob % ls my_data
backup.fasta
/export/home/rob %
```

UNIX Commands: **mv**

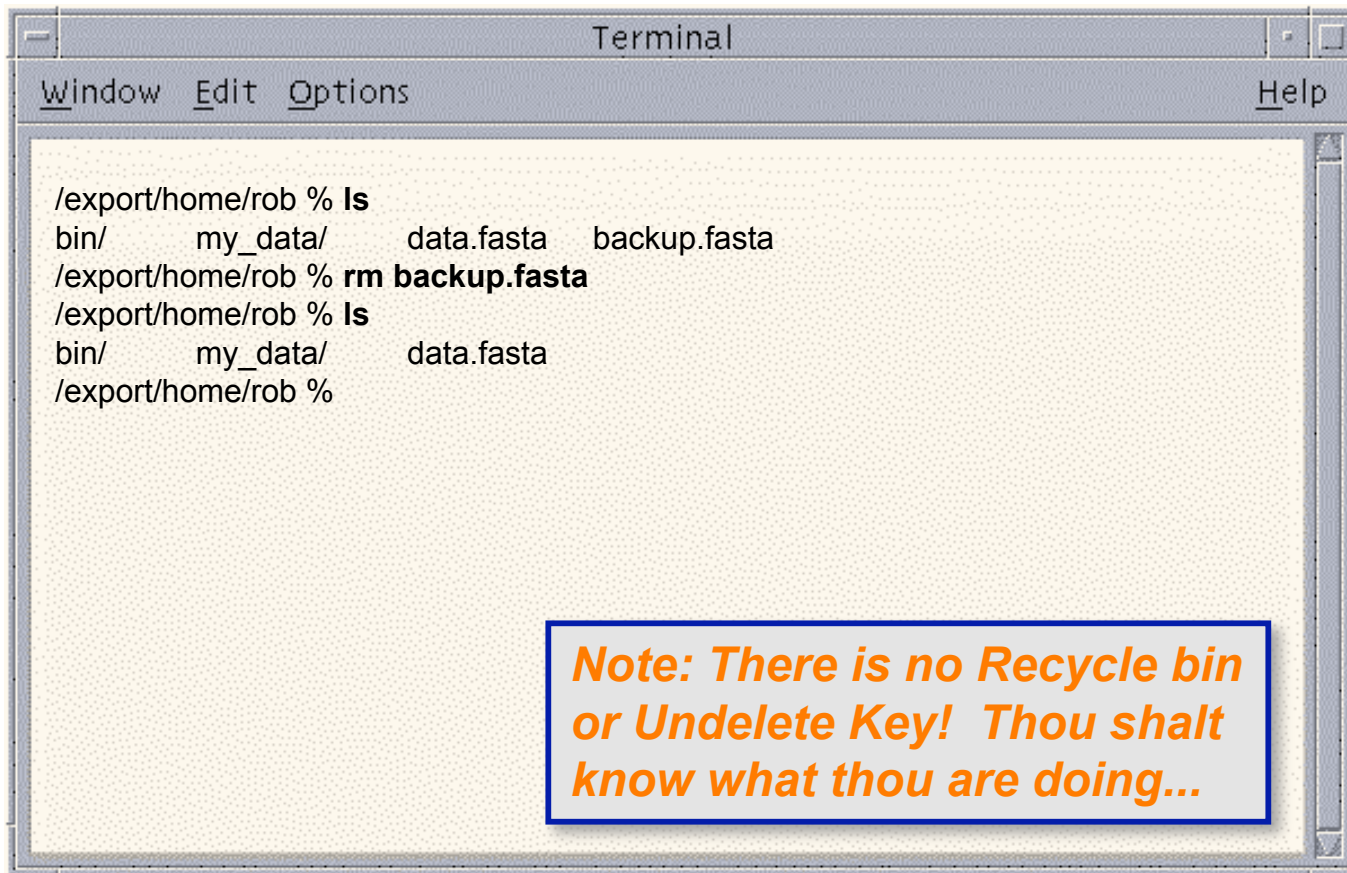
mv also renames files

A screenshot of a terminal window titled "Terminal". The window has a menu bar with "Window", "Edit", "Options", and "Help". The terminal content shows a series of commands and their outputs. First, the user runs "ls" in the directory "/export/home/rob", showing "bin/", "my_data/", "data.fasta", and "backup.fasta". Then, the user runs "mv backup.fasta my_data". Next, the user runs "ls" again, showing "bin/", "my_data/", and "data.fasta". Then, the user runs "ls my_data", showing "backup.fasta". Finally, the user runs "mv data.fasta Dec15.fasta". The last command shown is "ls" in the directory "/export/home/rob", which shows "bin/", "my_data/", and "Dec15.fasta".

```
/export/home/rob % ls
bin/      my_data/   data.fasta backup.fasta
/export/home/rob % mv backup.fasta my_data
/export/home/rob % ls
bin/      my_data/   data.fasta
/export/home/rob % ls my_data
backup.fasta
/export/home/rob % mv data.fasta Dec15.fasta
/export/home/rob % ls
bin/      my_data/   Dec15.fasta
/export/home/rob %
```

UNIX Commands: **rm**

rm deletes files - *permanently*.



A terminal window titled "Terminal" with a menu bar containing "Window", "Edit", "Options", and "Help". The terminal shows the following commands and output:

```
/export/home/rob % ls
bin/      my_data/    data.fasta  backup.fasta
/export/home/rob % rm backup.fasta
/export/home/rob % ls
bin/      my_data/    data.fasta
/export/home/rob %
```

Note: There is no Recycle bin or Undelete Key! Thou shalt know what thou are doing...

UNIX Commands: **rm**

rm options.

rm -i : interactive, ask for confirmation

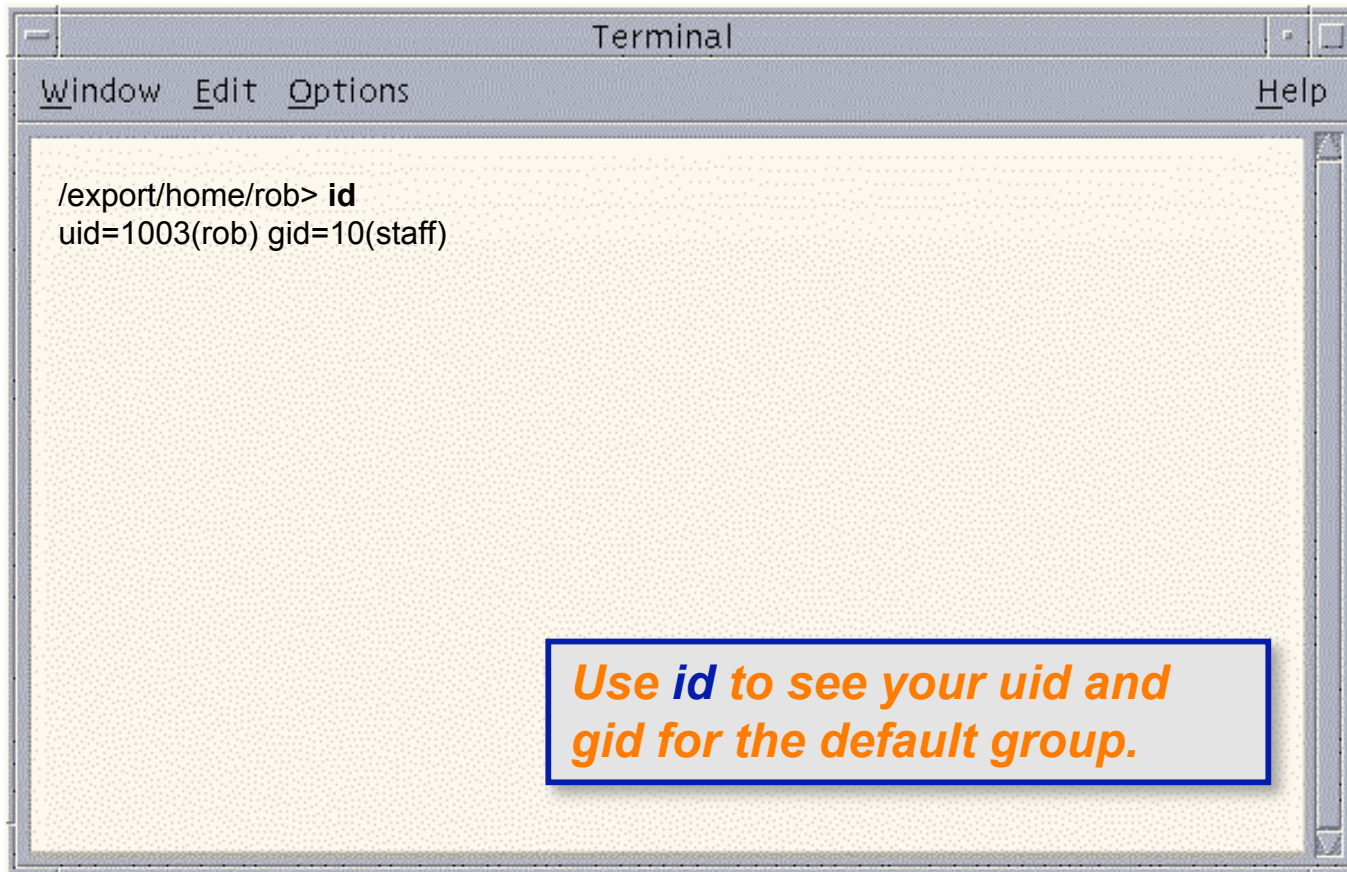
rm -r : recursively remove directories

rm -f : override lack of write permission

UNIX Commands: **head** & **tail**

- head displays the *top* of a file
 - **head -n** displays the top **n** lines
 - default is 10
- tail displays the *bottom* of a file
 - **tail -n** displays the bottom **n** lines
 - default is 10
 - **tail +n** displays the file starting at line **n**

UNIX Commands: **id**

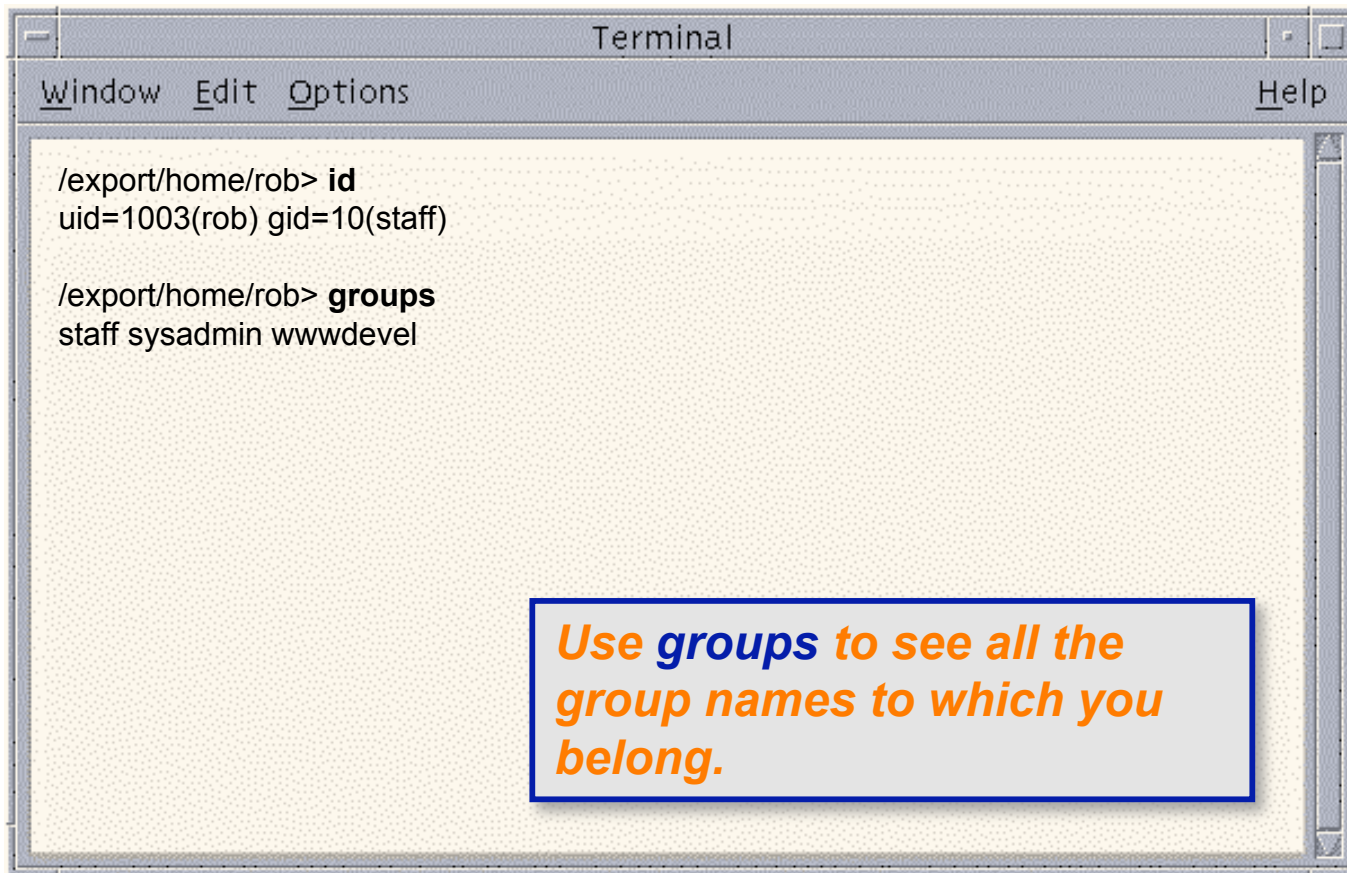


A terminal window titled "Terminal" with a menu bar containing "Window", "Edit", "Options", and "Help". The terminal shows the command `id` being executed at the prompt `/export/home/rob>`. The output is `uid=1003(rob) gid=10(staff)`. A blue-bordered box with orange text is overlaid on the bottom right of the terminal area.

```
/export/home/rob> id
uid=1003(rob) gid=10(staff)
```

*Use **id** to see your uid and gid for the default group.*

UNIX Commands: **groups**



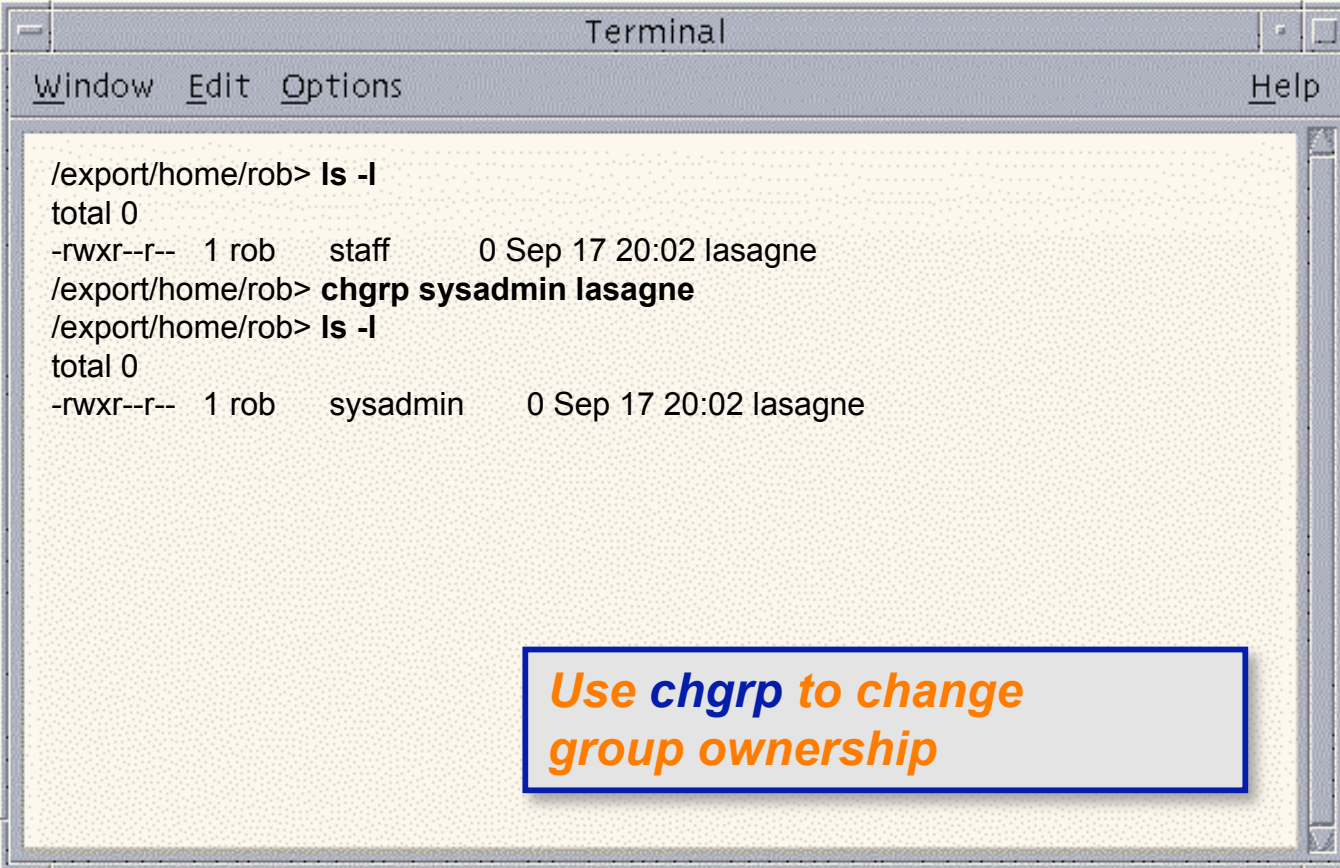
A terminal window titled "Terminal" with a menu bar containing "Window", "Edit", "Options", and "Help". The terminal shows the following commands and their outputs:

```
/export/home/rob> id
uid=1003(rob) gid=10(staff)

/export/home/rob> groups
staff sysadmin wwwdevel
```

An orange callout box with a blue border is positioned in the lower right of the terminal window, containing the text: *Use **groups** to see all the group names to which you belong.*

UNIX Commands: **chgrp**



A terminal window titled "Terminal" with a menu bar containing "Window", "Edit", "Options", and "Help". The terminal shows the following commands and output:

```
/export/home/rob> ls -l
total 0
-rwxr--r--  1 rob    staff      0 Sep 17 20:02 lasagne
/export/home/rob> chgrp sysadmin lasagne
/export/home/rob> ls -l
total 0
-rwxr--r--  1 rob    sysadmin   0 Sep 17 20:02 lasagne
```

A callout box with a blue border and orange text is positioned in the bottom right of the terminal window, containing the text: "Use **chgrp** to change group ownership".

UNIX Commands: **find**

- At its simplest, find searches the filesystem for files whose name matches a specific pattern
- However, it can do a lot more and is one of the most useful commands in Unix (as it can find specific files and then perform operations on them)
- Here is a simple example:

```
> ls
```

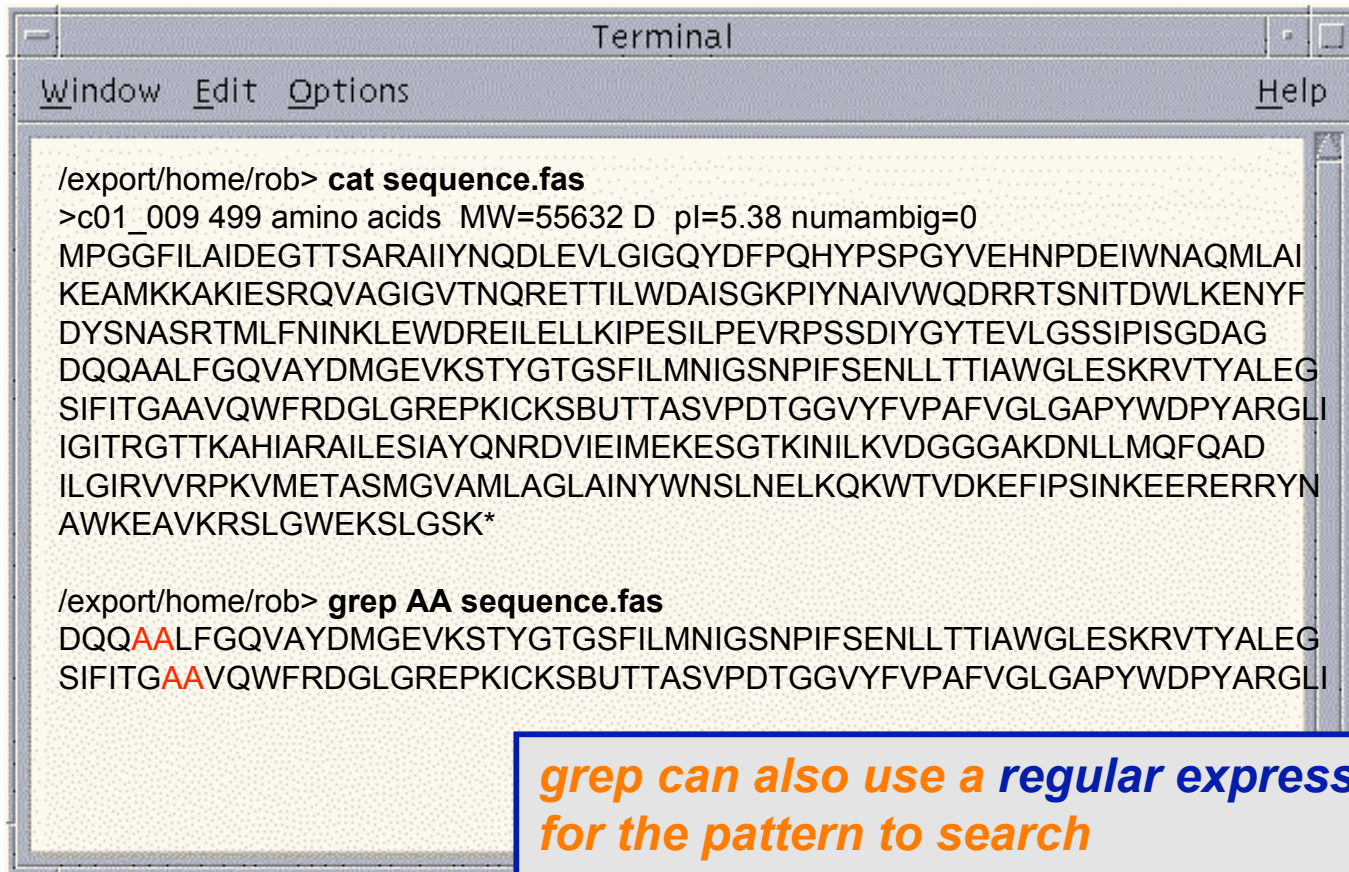
```
dir1  foo  foo2
```

```
> find . -name foo -print
```

```
./foo
```

UNIX Commands: **grep**

grep extracts lines from a file that match a given string or pattern



```
Terminal
Window Edit Options Help

/export/home/rob> cat sequence.fas
>c01_009 499 amino acids MW=55632 D pl=5.38 numambig=0
MPGGFILAIDEGTTSARAIYNQDLEVLGIGQYDFPQHYPSPGYVEHNPDEIWNAQMLAI
KEAMKKAKIESRQVAGIGVTNQRETTILWDAISGKPIYNAIVWQDRRTSNITDWLKENYF
DYSNASRTMLFNINKLEWDREILLELKIPESILPEVRPSSDIYGYTEVLGSSIPISGDAG
DQQAALFGQVAYDMGEVKSTYGTGSFILMNIGSNPIFSENLLTTIAWGLESKRVTYALEG
SIFITGAAVQWFRDGLGREPKICKSBUTTASVPDTGGVYFVPAFVGLGAPYWDPYARGLI
IGITRGTTKAHIAIRAILESIAYQNRDVIEIMEKESGTKINILKVDGGGAKDNLLMQFQAD
ILGIRVVRPKVMETASMGVAMLAGLAINYWNSLNELKQKWTVDKEFIPSINKEERERRYN
AWKEAVKRSLGWEKSLGSK*

/export/home/rob> grep AA sequence.fas
DQQAAALFGQVAYDMGEVKSTYGTGSFILMNIGSNPIFSENLLTTIAWGLESKRVTYALEG
SIFITGAAVQWFRDGLGREPKICKSBUTTASVPDTGGVYFVPAFVGLGAPYWDPYARGLI
```

grep can also use a regular expression for the pattern to search

Regular Expressions

- In addition to grep, a number of Unix commands support the use of *regular expressions* to describe patterns:
 - sed
 - awk
 - perl
- General search pattern characters:
 - Any character (*except a metacharacter*) matches itself
 - “.” matches any character except a newline
 - “*” matches zero or more occurrences of the single preceding character
 - “+” matches one or more of the preceding character
 - “?” matches zero or one of the preceding character
- Additional special characters:
 - “()” parentheses are used to quantify a sequence of characters
 - “|” works as an OR operator
 - “{}” braces are used to indicate ranges in the number of occurrences

Regular Expressions

- If you really want to match a period '.', you need to escape it with a backslash "\."

Regex	Matches	Does not match
a.b	axb	abc
a\.b	a.b	axb

Regular Expressions

- A *character class*, also called a character set can be used to match only one out of several characters
- To use, simply place the characters you want to match between square brackets []
- You can use a hyphen inside a character class to specify a range of characters
- Placing a caret (^) after the opening square bracket will negate the character class. The result is that the character class will match any character that is *not* in the character class
- Examples:
 - [abc] matches a *single* a b or c
 - [0-9] matches a *single* digit between 0 and 9
 - [^A-Za-z] matches a single character as long as it is not a letter

Regular Expressions

- Since certain character classes are used often, a series of shorthand character classes are available for convenience:

`\d` a digit. eg [0-9]

`\D` a non-digit, eg. [^0-9]

`\w` a word character (matches letters and digits)

`\W` a non-word character

`\s` a whitespace character

`\S` a non-whitespace character

Regular Expressions

- More shorthand classes are available for *matching boundaries*:

[^] the beginning of a line

^{\$} the end of a line

^{\b} a word boundary

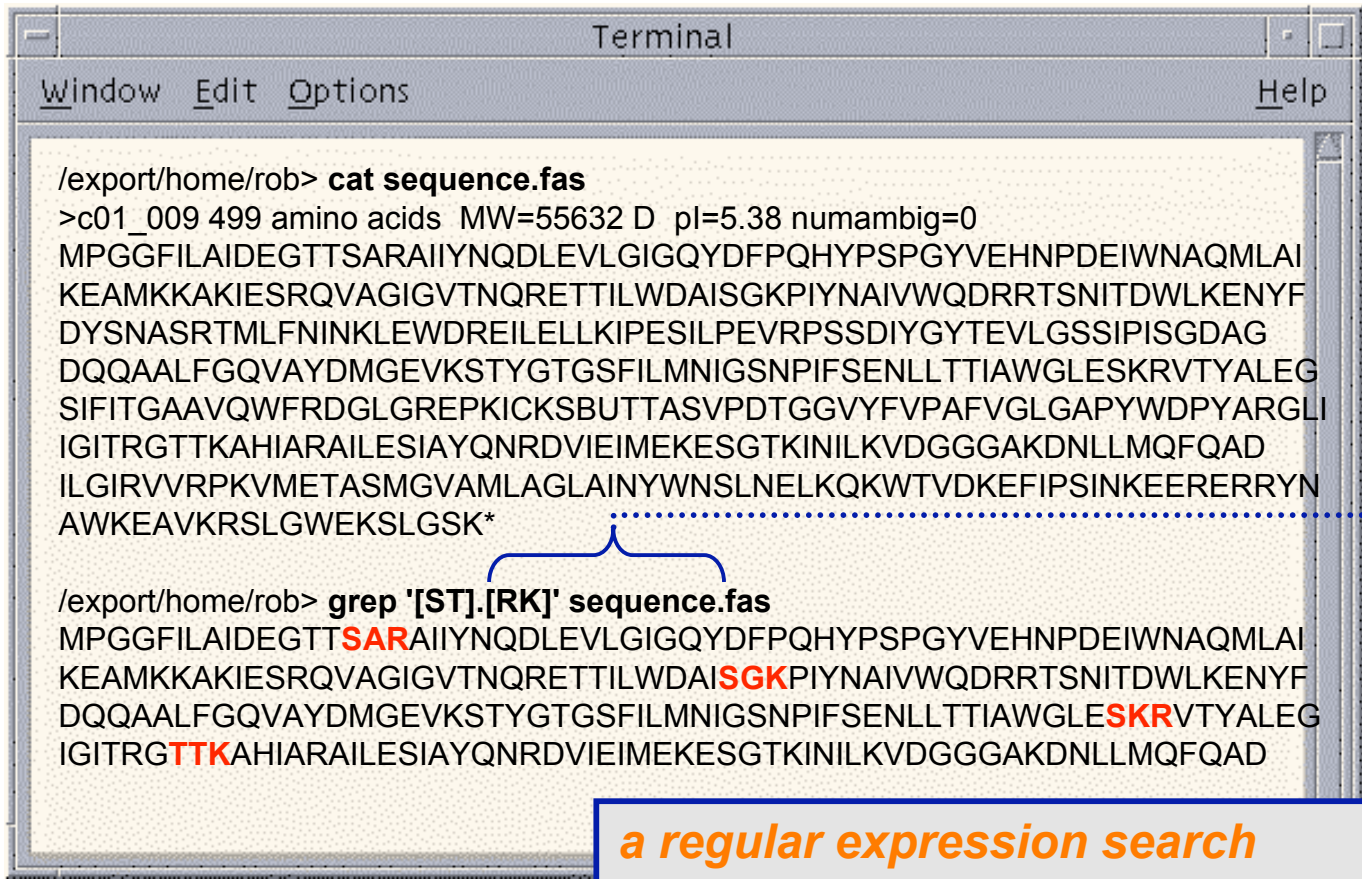
^{\B} a non-word boundary

Regular Expressions Examples

- “notice” a string that has the text "notice" in it
- “F.” matches an “F” followed by any character
- “a.b” matches “a” followed by any 1 char followed by “b”
- “^The” matches any string that starts with "The"
- “oh boy\$” matches a string that ends in the substring "oh boy";
- “^abc\$” matches a string that starts and ends with "abc" -- that could only be "abc" itself!
- “ab*” matches an “a” followed by zero or more “b”s ("a", "ab", "abbb", etc.)
- “ab+” similar to previous, but there's at least one “b” ("ab", "abbb", etc.)
- “(b|cd)ef” matches a string that has either "bef" or "cdef"
- “a(bc)*” matches an “a” followed by zero or more copies of the sequence "bc"
- “ab{3,5}” matches an “a” followed by three to five “b”s ("abbb", "abbbb", or "abbbbb")
- “[Dd][Aa][Vv][Ee]” matches "Dave" or "dave" or "dAVE“, does not match "ave" or "da"

UNIX Commands: **grep**

grep extracts lines from a file that match a given string or pattern



A terminal window titled "Terminal" with a menu bar containing "Window", "Edit", "Options", and "Help". The terminal shows the following commands and output:

```
/export/home/rob> cat sequence.fas
>c01_009 499 amino acids MW=55632 D pl=5.38 numambig=0
MPGGFILAIDEGTTSARAIYNQDLEVLGIGQYDFPQHYPSPGYVEHNPDEIWNAQMLAI
KEAMKKAKIESRQVAGIGVTNQRETTILWDAISGKPIYNAIVWQDRRTSNITDWLKENYF
DYSNASRTMLFNINKLEWDREILLELKIPESILPEVRPSSDIYGYTEVLGSSIPISGDAG
DQQAALFGQVAYDMGEVKSTYGTGSFILMNIGSNPIFSENLLTTIAWGLESKRVTYALEG
SIFITGAAVQWFRDGLGREPKICKSBUTTASVPDTGGVYFVPAFVGLGAPYWDPYARGLI
IGITRGTTKAHIAIRAILESIAYQNRDVIEIMEKESGTKINILKVDGGGAKDNLLMQFQAD
ILGIRVVRPKVMETASMGVAMLAGLAINYWNSLNELKQKWTVDKEFIPSINKEERERRYN
AWKEAVKRSLGWEEKSLGSK*
```

A blue bracket is drawn under the pattern `'[ST].[RK]'` in the second command. A dotted blue line extends from the right side of the bracket to a blue arrow pointing at a box.

```
/export/home/rob> grep '[ST].[RK]' sequence.fas
MPGGFILAIDEGTTSARAIYNQDLEVLGIGQYDFPQHYPSPGYVEHNPDEIWNAQMLAI
KEAMKKAKIESRQVAGIGVTNQRETTILWDAISGKPIYNAIVWQDRRTSNITDWLKENYF
DQQAALFGQVAYDMGEVKSTYGTGSFILMNIGSNPIFSENLLTTIAWGLESKRVTYALEG
IGITRGTTKAHIAIRAILESIAYQNRDVIEIMEKESGTKINILKVDGGGAKDNLLMQFQAD
```

The matched substrings are highlighted in red: **SAR**, **SGK**, **SKR**, and **TTK**.

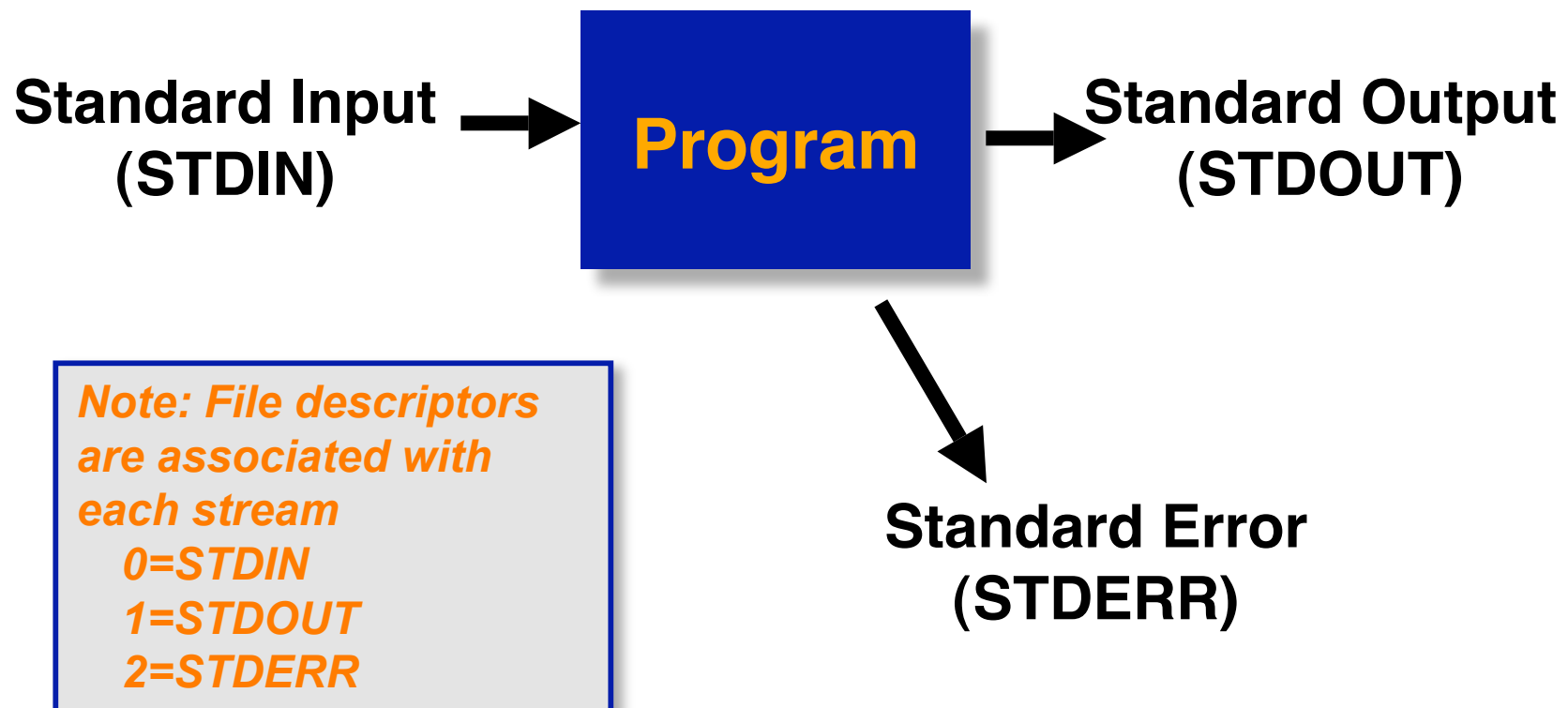
a regular expression search

Interacting with the Shell

Running a Unix Program

- Typically, you type in the name of a program and some command line options
- The shell reads this line, finds the program and runs it, feeding it the options you specified
- The shell establishes 3 separate I/O *streams*:
 - Standard Input
 - Standard Output
 - Standard Error

Programs and Standard I/O



Defaults for I/O

- When a shell runs a program for you:
 - standard input is your keyboard
 - standard output is your screen or window
 - standard error is your screen or window
- If standard input is your keyboard, you can type stuff in that goes to a program
- To end the input you press Ctrl-D (^D) on a line by itself, this ends the input *stream*
- The shell is a program that reads from standard input
- Any idea what happens when you give the shell ^D?

UNIX: Shell Flavors

- There are two main ‘flavors’ of shells:
 - Bourne created what is now known as the standard shell: “**sh**”, or “**bourne** shell”. Its syntax roughly resembles Pascal. Its derivatives include “**ksh**” (“korn shell”) and now, the most widely used, “**bash**” (“bourne again shell”)
 - One of the creators of the C language implemented a shell to have a “C-programming” like syntax. This is called “**cs**h” or “C-shell”. Today’s most widely used form is the very popular “**tcsh**”
- Shells can run interactively or as a *shell script*

Customization

- Each shell supports some customization.
 - user prompt settings
 - environment variable settings
 - aliases
- The customization takes place in *startup* files which are read by the shell when it starts up
 - Global files are read first - these are provided by the system administrators (eg. /etc/profile)
 - Local files are then read in the user's HOME directory to allow for additional customization

Shell Startup Files

sh,ksh:

~/ .profile

bash:

~/ .bash_profile

~/ .bash_login

~/ .profile

~/ .bashrc

~/ .bash_logout

csh:

~/ .cshrc

~/ .login

~/ .logout

tcsh:

~/ .tshrc

~/ .cshrc

~/ .login

~/ .logout

Note: on TACC production systems, we provide an alternative location for customization files to avoid over-riding system defaults:

***BASH:** ~/ .profile_user*

***CSH/TCSH:** ~/ .login_user*

~/ .cshrc_user

Wildcards for Filename Abbreviation

- When you type in a command line the shell treats some characters as special (*metacharacters*)
- These special characters make it easy to specify filenames
- The shell processes what you give it, using the special characters to replace your command line with one that includes a bunch of file names

The special character *

- “*” matches anything.
- If you give the shell “*” by itself (as a command line argument), the shell will remove the * and replace it with all the filenames in the current directory.
- “a*b” matches all files in the current directory that start with **a** and end with **b**.
- This looks like regular expressions but isn’t quite.

Understanding *

- The **echo** command prints out whatever you tell it:

```
> echo hi  
hi
```

```
> ls  
dir1  foo  foo2
```

- What will the following command do?

```
> echo *  
dir1 foo foo2
```

Shell Stream Redirection

- A very powerful function in Unix is redirection for input and output:
 - The shell can attach things other than your *keyboard* to *standard input (stdin)*
 - A file (the contents of the file are fed to a program as if you typed it) - common in scientific programming
 - A pipe (the output of another program is fed as input as if you typed it)
 - The shell can attach things other than your *screen* to *standard output (stderr)*
 - A file (the output of a program is stored in file)
 - A pipe (the output of a program is fed as input to another program)

Stream Redirection

- To tell the shell to store the *output* of your program in a file, follow the command line for the program with the “>” character followed by the filename:

```
ls > lsout
```

- The command above will create a file named `lsout` and place the output of the `ls` command in the file

Stream Redirection

- To have the shell get standard *input* from a file, use the “<” character:

```
sort < nums
```

- The command above would sort the lines in the file **nums** and send the result to *stdout*
- Beauty of redirection is that you can do both forms together:

```
sort < nums > sortednums
```

Modes of Output Redirection

- There are two modes of output redirection:
 - “>” the create mode
 - “>>” the append mode
- For example:
 - the command `ls > foo` will create a new file named foo (deleting any existing file named foo).
 - if you use “>>” instead, the output will be appended to foo:

```
ls /etc >> foo
ls /usr >> foo
```

Stream Redirection

- Many commands send error messages to *standard error* (*stderr*) which is different from *stdout*.
- However, the “>” output redirection only applies to *stdout* (not *stderr*)
- To redirect *stderr* to a file you need to know what shell you are using:
 - BASH
 - “2>” redirects *stderr* (eg. `ls foo blah gork 2> erroroutput`)
 - “&>” redirects *stdout* and *stderr* (eg. `ls foo &> /dev/null`)
 - TCSH
 - “>&” merges *stdout* and *stderr* and sends to a file:
`ls foo blah >& saveboth`
 - “>>&” merges *stdout* and *stderr* and appends to a file:
`ls foo blah >>& saveboth`

Example of stderr/out

```
[albook:~/tst] %% cat errout.c
```

```
#include <stdlib.h>
```

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

```
int main()
```

```
{
```

```
    fprintf(stdout,"a1\n");
```

```
    fprintf(stderr,"b2\n");
```

```
    return 0;
```

```
}
```

```
[albook:~/tst] %% cat erroutf.f
```

```
    program errout
```

```
    write(6,*) "a1"
```

```
    write(0,*) "b2"
```

```
end program
```

```
[albook:~/tst] %% cc -o errout errout.c
```

```
[albook:~/tst] %% errout
```

```
a1
```

```
b2
```

```
[albook:~/tst] %% errout > what.out
```

```
b2
```

```
[albook:~/tst] %% cat what.out
```

```
a1
```

```
[albook:~/tst] %% errout 1> out.out 2> err.out
```

```
[albook:~/tst] %% cat out.out
```

```
a1
```

```
[albook:~/tst] %% cat err.out
```

```
b2
```

```
[albook:~/tst] %% errout > all.out 2>&1
```

```
[albook:~/tst] %% cat all.out
```

```
b2
```

```
a1
```

```
[albook:~/tst] %% errout &> all.out
```

```
[albook:~/tst] %% cat all.out
```

```
b2
```

```
a1
```

Note: this only works this way in sh/bash



References/Acknowledgements

- National Research Council Canada (Rob Hutten, Canadian Bioinformatics Resource)
- *Intro. to Unix*, Dave Hollinger, Rensselaer Polytechnic Institute
- **Unix in a Nutshell**, A. Robbins, O'Reilly Media, 2006.
- Regular expression info (<http://www.regular-expressions.info/reference.html>)