Data Analysis in Geophysics
ESCI 7205

Class 3

Bob Smalley

Basics of Unix commands

UNIX is a four letter word

"Unix is user friendly –

It's just picky about who it's friends are…"

-- Unknown, seen in .sigs around the world

Manipulating files

`paste`:
concatenate files with each file a new column; when used on a single file, it dumps the entire file contents to the screen.

(**`cat`** sticks the files together one after the other. **`paste`** puts them together a line at a time. Each line N of the output file from paste is made up of the lines N of the input files.)

Looking at files

`head –nX`:
prints the first X number of lines to the screen;
default is 10 lines if -n is not specified.

`tail –nX`:
prints the last X number of lines to the screen;
default is 10 lines if -n is not specified.

Piping and Redirect

Input and output on the command line are controlled by the |, >, <, and ! Symbols.

| : pipe function; sends the output from command on left side as input to the command on the right side.

(We have seen these actions already.)

Piping and Redirect

Example pipe

```
% ls | head -n5
29-sadvf1
29-sadvf2
2meas.sh.out.txt
3132.dat
31all32new.trk
%
```

Piping and Redirect

“>” redirects standard output (screen) to a specific file*

```
% ls | head -n5 > directory.list
% more directory.list
29-sadvf1
29-sadvf2
2meas.sh.out.txt
3132.dat
31all32new.trk
```

* In tcsh, this will not overwrite (clobber) a pre-existing file with the same name. In the bash shell, the > overwrites (clobbers) any pre-existing file with no warning!

Piping and Redirect

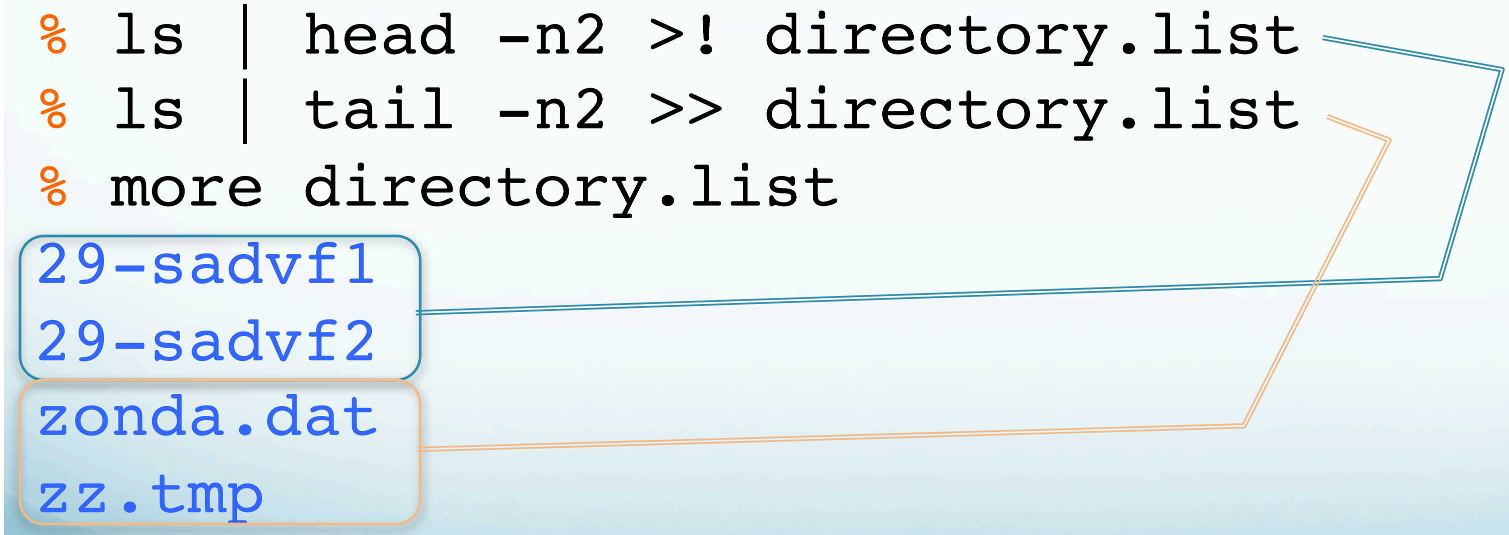
>! : redirects standard output (screen output) to a specific file and overwrite (clobber) the file if it already exists *

```
% ls | head –n5 >! directory.list
% more directory.list
29-sadvf1
29-sadvf2
2meas.sh.out.txt
3132.dat
31all32new.trk
```

*This syntax is specific to tcsh, your default CERI shell; in bash this will put the output into a file named “ ! ”!

Piping and Redirect

>> : redirects and concatenates standard output (screen output) to the end of a specific (existing) file



```
% ls | head -n2 >! directory.list
% ls | tail -n2 >> directory.list
% more directory.list
29-sadvf1
29-sadvf2
zonda.dat
zz.tmp
```

Piping and Redirect

< : redirects input from Standard input to the file on right of the less-than sign to be used as input to command on the left

```
% head –n1 < suma1.hrdpicks
51995   31410273254    30870   958490
```

Copying files & directories

`cp:`

copy files

`cp –r:`

copy directory and all files & subdirectories within it (recursive copy)

Copying files & directories

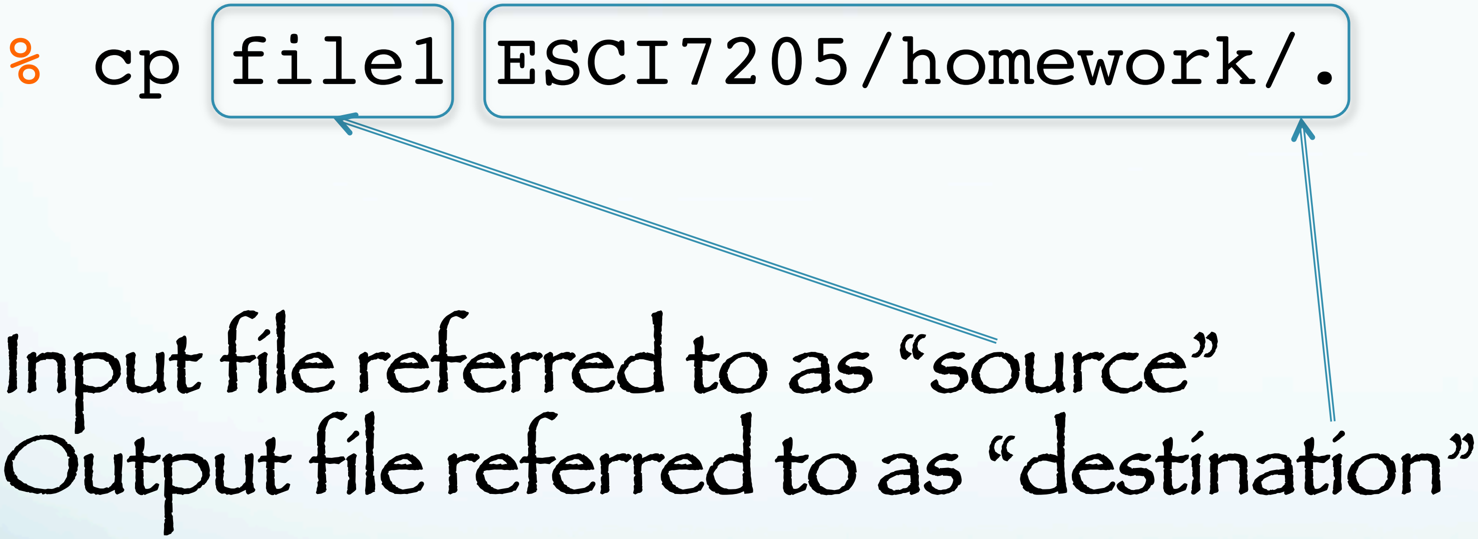
```
% cp file1 ESCI7205/homework/HW1
```

Makes a copy with a new name – "HW1" in the directory "ESCI7205/homework"

```
% cp file1 ESCI7205/homework/.
```

Makes a copy with the same name (file1), which is specified by the dot "." (period) to save typing, in the new directory.

Some jargon



```
% cp file1 ESCI7205/homework/.
```

Input file referred to as "source"
Output file referred to as "destination"

Moving files & directories
mv: move files or directories

```
% mv file1 file2 ESCI7205/HW/.
```

Moves file1 and file2 to new directory (relative) ESCI7205/HW with same names (indicated by the "**.**").

Move differs from copy in that it removes the original file, you only have 1 copy of it when done.

Moving files & directories

mv: move files or directories

```
% mv file1 ESCI7205/HW/HW1
% mv file2 ESCI7205/HW/HW2
```

If you want to change the names when you move them, you have to specify each new file name (do them one at a time)

Renaming files & directories

(you should have been able to figure this out after the last two slides)

Uses a side-effect of move!!!

```
% mv file1 HW1
% mv file2 HW2
```

There is NO RENAME command.
(We consistently see this kind of inconsistent logic in Unix.)

Linking files & directories

`ln –s:`

creates a symbolic link between two files.

This makes the file show up somewhere(the target, can be a new name in the same directory or the same name in another directory), but the file really only exists in the original place.

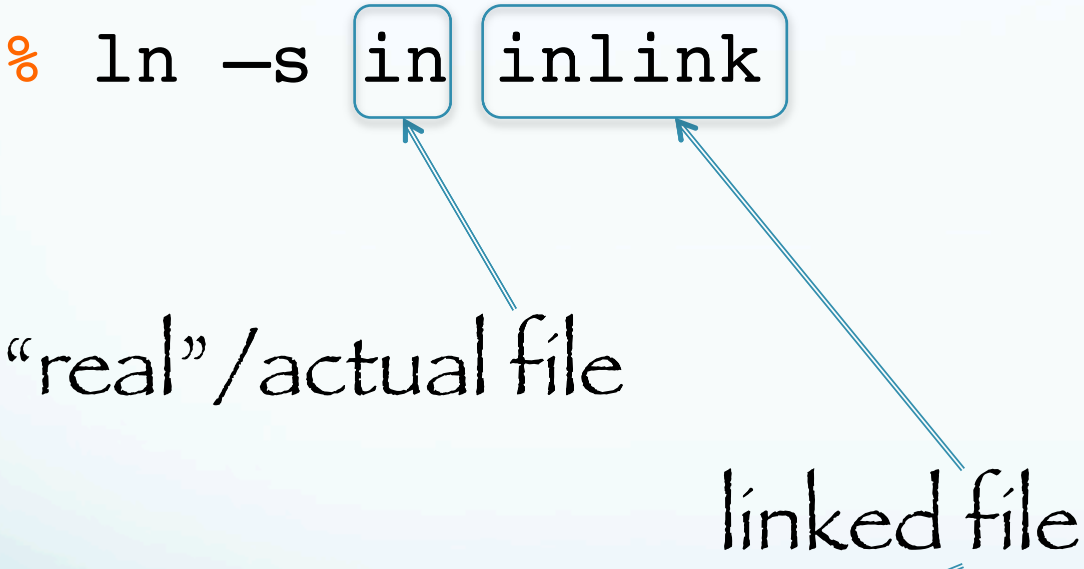
(equivalent to a file alias in OSX or shortcut in Windows).

Try reading the man page –

```
LN(1)                 BSD General Commands Manual                LN(1)

NAME
     link, ln -- make links

SYNOPSIS
     ln [-Ffhinsv] source_file [target_file]
     ln [-Ffhinsv] source_file ... target_dir
     link source_file target_file

DESCRIPTION
     The ln utility creates a new directory entry (linked file)
which has the same modes as the original file.  It is useful for
maintaining multiple copies of a file in many places at once
without using up storage for the ``copies''; instead, a link
``points'' to the original copy.  There are two types of links;
hard links and symbolic links.  How a link ``points'' to a file
is one of the differences between a hard and symbolic link.
```

Linking files & directories

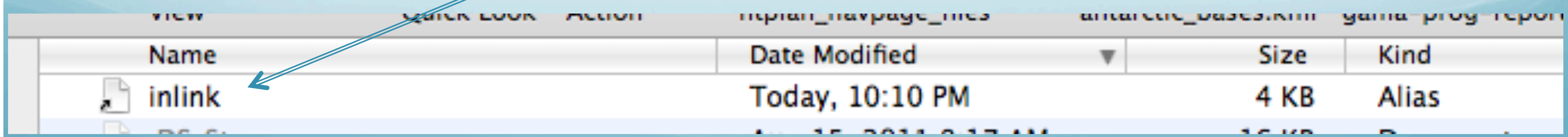
Two kinds of link - symbolic and hard. Only root can make hard links so don't worry about them.



```
% ln –s in inlink
```

"real"/actual file

linked file



Name	Date Modified	Size	Kind
inlink	Today, 10:10 PM	4 KB	Alias

Linking files & directories

Doing an ls command in the directory with the alias produces the following

```
$ ls -l in*
-rw-r--r--@ 1 smalley staff 69 Apr 26 2010 in
lrwxr-xr-x  1 smalley staff  2 Sep  2 22:10 inlink -> in
```

The leading “`l`” in the long `ls` output says the file/filename in that line is a link.

It shows which file it is linked to.

Linking files & directories

This allows us to “have” the file in more than one place.

We can therefore access it locally from the directory where it is a symbolic link.

Introduction to wildcards.

Wildcards are essential when dealing with almost anything in terms of text processing. (Looking for/Managing files from the command line is text processing.)

They are a subset of regular expressions, an essential (i.e. esoteric and difficult) Unix feature.

Wildcards

Wildcards allow you to match multiple instances of characters/numbers in file or directory names

They can be used in combination with almost all Unix commands

Wildcards are essential when dealing with large amounts of geophysical data

Introduction to wildcards.

Example

Say I want to find all the files in the working directory that begin with the letter "**a**".

(lower case only since Unix is case sensitive.)

Start out with the ls command

How do we specify we want all combinations of all characters following the "**a**"?
We use a wildcard.

```
% ls a*
```

The asterisk "*" wildcard means match a string with any number of any character (including none, so will match a file "**a**").

Try it ---

```
> ls a*
a.out                               antex.sh
antarctic sun panorama 3x.ai        atantest.f
antarctic sun panorama.125.jpg      awk
antarctic sun panorama.25.jpg       az_map
antarctic sun panorama.ai           az_map.ps
antarctic sun panorama.jpg

adelitst:
aadeli.ini       adelitst.sh       jessai          pessai
ADELI.MESSAGES   eessai            kcnusc.pal      PLOT1
ADELI.MINMAX     iessai            oessai          tempi

arc2gmtstuff:
arcgmt.README   arcgmt.tar     arcgmt_ai      arcgmt_av
>
```

Probably not what you wanted though – it lists files starting with "a" and then goes recursively through all directories that start w/ "a".

Try it ---

```
> ls –d a*
a.out                           antex.sh
antarctic sun panorama 3x.ai    atantest.f
antarctic sun panorama.125.jpg  awk
antarctic sun panorama.25.jpg   az_map
antarctic sun panorama.ai       az_map.ps
antarctic sun panorama.jpg
>
```

Flag **–d** says do not go recursively through all directories (that start w/ "**a**").

Use man page to figure this out.

(As part of the regular expression feature of Unix) wildcards can be used in combination with almost all Unix commands.

Wildcards

“*” – asterisk - matches zero or more characters or numbers.

Combining/multiple use of wildcards.

Find all files in local subdirectory SEIS that begin with the letter “f” and also have the string “.BHZ.” in their file name.

```
%ls SEIS/f*.BHZ.*
SEIS/filt.HIA.BHZ.SAC  SEIS/filt.WMQ.BHZ.SAC
```

“?” – question mark - matches a single character or number.

Find all files in local subdirectory SEIS that have the name “**HIA.BH**” plus some single letter (the **?**) plus a “**.**” and then plus anything (the *****).

```
% ls  SEIS/HIA.BH?.*
SEIS/HIA.BHE.SAC       SEIS/HIA.BHN.SAC
SEIS/HIA.BHZ.SAC
```

Wildcards

“[   ]” – brackets - used to specify a set or range of characters or numbers rather than all possible characters or numbers.

Find all files in local subdirectory SEIS that have the name “**HIA.BH**” plus one of E, N or Z (the stuff in brackets) plus a “.” and then plus anything (the *).

```
% ls  SEIS/HIA.BH[E,N,Z].*
SEIS/HIA.BHE.SAC           SEIS/HIA.BHZ.SAC
SEIS/HIA.BHN.SAC
```

Wildcards

Find all files in all local subdirectories (the first *) that have the string “`HIA`” in the name plus anything (the second *) plus the characters “`198`” plus a single character in the range 0-9 then plus anything (the third and last *).

```
% ls  */HIA*198[0-9]*
795/HIA.BHZ.D.1988.041:07.18.30
799/HIA.BHZ.D.1988:14:35:27.00
812/HIA.BHZ.D.1988:03:43:49.00
813/HIA.BHZ.D.1988.362:13.58.59
814/HIA.BHZ.D.1989.041:17.07.43
```

Some random stuff
Control-characters(CTRL-characters)

ctrl-s freezes the screen and stops any display on the screen from continuing (equivalent to a no-scroll key) (sometimes takes a moment to work)

ctrl-q un-freezes the screen and lets screen display

continue ctrl-c interrupts a running program

ctrl-\ same as ctrl-c but stronger (used when terminal doesn't respond)

Some random stuff
Control-characters (CTRL-characters)

ctrl-z suspends a running program (use the fg command to continue the program)

ctrl-h deletes last character typed

ctrl-w deletes last word typed

ctrl-u deletes last line typed

Some random stuff
Control-characters(CTRL-characters)

ctrl-r redraws last line typed ctrl-d ends text input for many UNIX programs, including mail and write.

(http://web.cecs.pdx.edu/~rootd/catdoc/guide/TheGuide_38.html)

Some random stuff

A note on the book

As the book was not written for the CERI system, some of the files it refers to are not located where the book says they are.

What we have seen so far

Commands

```
cd
pwd
ls
mkdir
rmdir
rm
more
less
cat
paste
head
tail
cp
mv
Ln
echo
man
```

See this link for a list and description of many Unix commands

http://pcsplace.com/tech-list/ultimate-list-of-linux-and-unix-commands/

What we have seen so far

~~~~~~

Redirection

Pipes

Switches

Some special characters (~ \ . ..)

Wildcards (* ?)
~~~~~~

Man Pages

Basics of the Unix/Linux Environment

Using man pages

Layout

All man pages follow a common layout that is optimized for presentation on a simple ASCII text display (teletype), without any form of highlighting or font control.

Using man pages

Typical man page has following "headings":

SECTION
NAME
SYNOPSIS
DESCRIPTION
OPTIONS
OPERANDS
USAGE
(EXAMPLES)
ENVIRONMENT VARIABLES
EXIT STATUS
(FILES)
ATTRIBUTES
SEE ALSO
NOTES
(BUGS)

```
> man ls
Reformatting page.  Please Wait... done

User Commands                                              ls(1)

NAME
     ls - list contents of directory

SYNOPSIS
     /usr/bin/ls [-aAbcCdfFghilLmnopqrRstux1@] [file...]

     /usr/xpg4/bin/ls [-aAbcCdfFghilLmnopqrRstux1@] [file...]

DESCRIPTION
     For each file that is a directory, ls lists the contents  of
     the  directory.  For  each file that is an ordinary file, ls
     repeats its name and any other  information  requested.  The
     output is sorted alphabetically by default. When no argument
     is given, the current  directory  is  listed.  When  several
     arguments   are   given,  the  arguments  are  first  sorted
     appropriately, but file arguments appear before  directories
     and their contents.

     There are three major listing formats.  The  default  format
     for  output  directed  to  a  terminal  is multi-column with
     entries sorted down the columns. The -1 option allows single
     column  output and -m enables stream output format. In order
     to determine output formats for the -C, -x, and -m  options,
     ls  uses  an environment variable, COLUMNS, to determine the
     number of character positions available on one output  line.
     If  this  variable  is  not set, the terminfo(4) database is
     used to determine  the  number  of  columns,  based  on  the
     environment  variable,  TERM.  If this information cannot be
     obtained, 80 columns are assumed.

     The mode printed under the -l option consists of ten charac-
     ters. The first character may be one of the following:
```

SECTION

NAME

SYNOPSIS

DESCRIPTION

Using man pages

SECTION: The section of the manual. Includes command whose man page you requested.

```
User Commands                                  ls(1)
```

The `ls` commnad is in the "User Commands" section of the documentation/manual, which is section #1.

NAME: The name of the command or function, followed by a one-line description of what it does.

```
NAME
     ls - list contents of directory
```

Using man pages

SYNOPSIS

In the case of a command, you get a formal description of how to run it and what command line options it takes. For program functions, a list of the parameters the function takes and which header file contains its definition. For experienced users, this may be all the documentation they need.

Using man pages

SYNOPSIS (not so obvious)

Shows where command lives - **/usr/bin/** - (there are 2 versions available, depends on your path – more on paths later), plus …

```
SYNOPSIS
     /usr/bin/ls [-aAbcCdfFghilLmnopqrRstux1@] [file...]

     /usr/xpg4/bin/ls [-aAbcCdfFghilLmnopqrRstux1@] [file...]
```

Using man pages

SYNOPSIS (not so obvious)

...list of options

{[-aAbcCdfFghilLmnopqrRstux1@]}

the brackets { [ ] } signify that the stuff inside the brackets is optional, and ...

```
SYNOPSIS
     /usr/bin/ls [-aAbcCdfFghilLmnopqrRstux1@] [file...]

     /usr/xpg4/bin/ls [-aAbcCdfFghilLmnopqrRstux1@] [file...]
```

Using man pages

SYNOPSIS (not so obvious)

… finally, optionally (the brackets) a file name (file), that may be repeated an arbitrary number of times – the ellipses { . . . }.

```
SYNOPSIS
     /usr/bin/ls [-aAbcCdfFghilLmnopqrRstux1@] [file...]

     /usr/xpg4/bin/ls [-aAbcCdfFghilLmnopqrRstux1@] [file...]
```

Using man pages

Brackets – optional parameters.

File – filename.

Ellipses – repeat as necessary.

Using man pages

DESCRIPTION

A textual description of the functioning of the command or function.

Using man pages

DESCRIPTION

The DESCRIPTION can go on for a number of pages.

```
DESCRIPTION
     For each file that is a directory, ls lists the contents  of
     the  directory.  For  each file that is an ordinary file, ls
     repeats its name and any other  information  requested.  The
     output is sorted alphabetically by default. When no argument
     is given, the current  directory  is  listed.  When  several
     arguments   are   given,  the  arguments  are  first  sorted
     appropriately, but file arguments appear before  directories
     and their contents.

     There are three major listing formats.  The  default  format
```

This is where we find out what the first letters of the long `ls` format mean

```
The mode printed under the -l option consists of ten charac-
  ters. The first character may be one of the following:

  d     The entry is a directory.

  D     The entry is a door.

  l     The entry is a symbolic link.

  b     The entry is a block special file.

  c     The entry is a character special file.

  p     The entry is a FIFO (or "named pipe") special file.

  s     The entry is an AF_UNIX address family socket.

  -     The entry is an ordinary file.
```

etc.

Using man pages

OPTIONS

Specification of the command's options

```
OPTIONS
     The following options are supported:

     -a    Lists all entries, including those that begin  with  a
           dot (.), which are normally not listed.

     -A    Lists all entries, including those that begin  with  a
           dot  (.),  with the exception of the working directory
           (.) and the parent directory (..).

     -b    Forces printing of non-printable characters to  be  in
           the octal \ddd notation.
```

This can go on for pages also.

Using man pages

OPERAND

Describes the valid operands.

```
OPERANDS
     The following operand is supported:

     file  A path name of a file  to  be  written.  If  the  file
           specified  is  not found, a diagnostic message will be
           output on standard error.
```

Explains the operand is optional file name(s).

Using man pages

USAGE

Notes on usage (not examples).

```
USAGE
     See largefile(5) for the description of the behavior  of  ls
     when encountering files greater than or equal to 2 Gbyte ( 2
    **31 bytes).
```

Using man pages

EXAMPLES

Optionally (more like rarely) gives some examples.

```
EXAMPLES
     Example 3: Providing file information
     Another example of a command line is:

     example% ls -aisn

     This command provides information on  all  files,  including
     those  that  begin  with  a dot (a), the i-number-the memory
     address of the i-node associated with  the  file-printed  in
     the left-hand column (i); the size (in blocks) of the files,
     printed in the column to the right  of  the  i-numbers  (s);
     finally,  the  report is displayed in the numeric version of
     the long list, printing the UID (instead of user  name)  and
     GID  (instead  of  group  name)  numbers associated with the
     files.
     When the sizes of the files in a  directory  are  listed,  a
     total   count  of  blocks,  including  indirect  blocks,  is
     printed.
```

Using man pages

Followed by a bunch of other (mostly) esoteric stuff.

ENVIRONMENT VARIABLES (these can get you)

EXIT STATUS

FILES

ATTRIBUTES

(the following may be useful) SEE ALSO

NOTES

BUGS.

Shells

Basics of the Unix/Linux Environment

What is a shell?

As far as Unix is concerned, the shell is just another program.

As far as the user in concerned, it is the traditional command line user interface with the Unix operating system…it interprets your typing.

What is a shell?

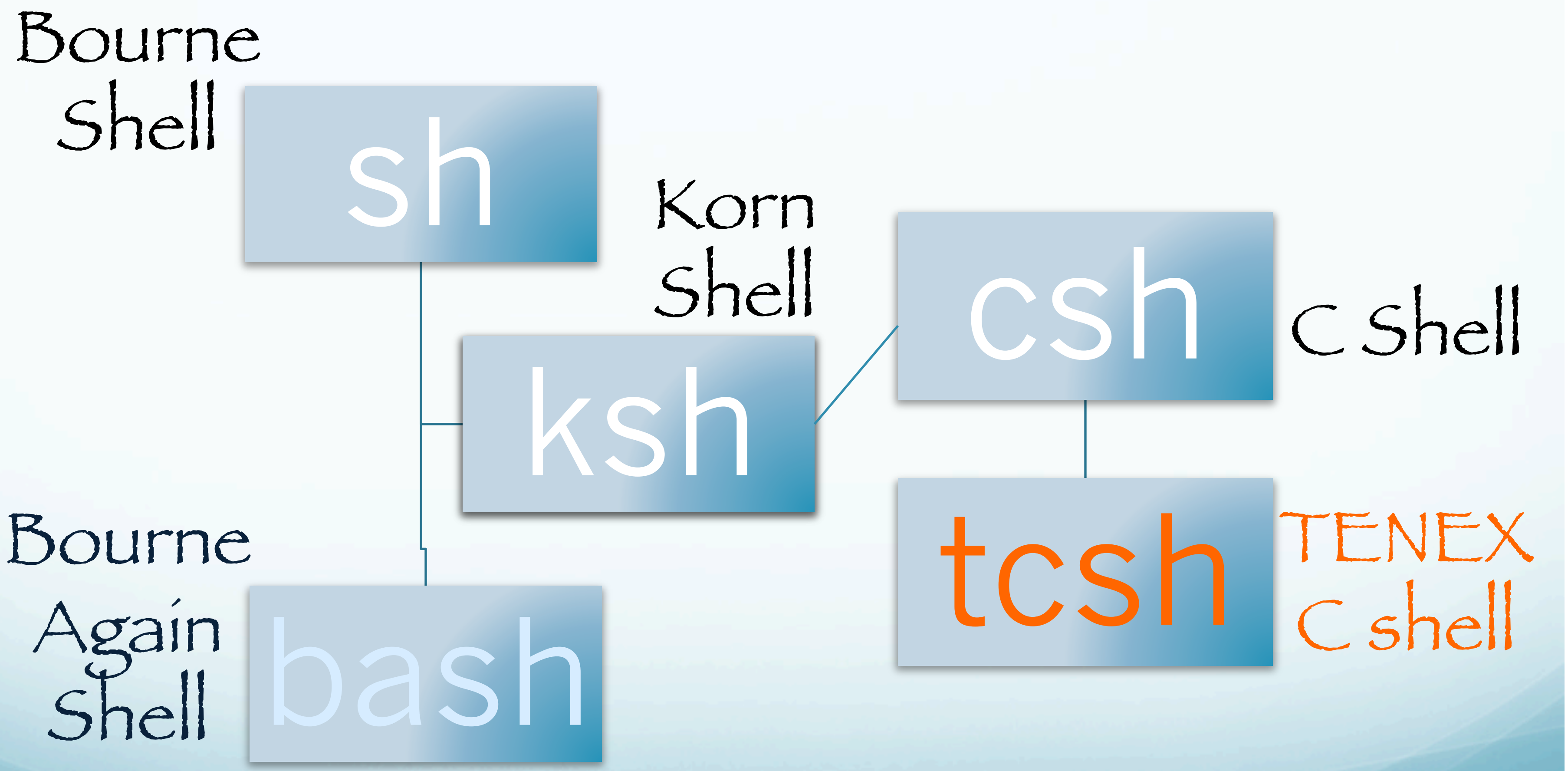
Just as there are many flavors of Unix and Unix-like systems, there are many types of shells.

If you don't like any of the shells in existence, this is Unix – write your own!

Common shells

Bourne Shell	sh
Bourne Again Shell (current default on MAC OS X)	bash
C Shell	csh
TENEX C Shell (This is the default shell at CERI)	tcsh
Korn Shell (mix between two shell families above)	ksh

Common shells



sh

Bourne shell

The original Unix shell.

Pro: Flexible and powerful scripting shell.

Con: Not interactive or particularly user friendly.

csh

C shell

designed for the BSD Unix system.

syntax closely follows C programming.

Pro: easy for C programmers to learn and comes with many interactive features such as file completion, aliases, history.

Con: not as flexible or powerful a scripting language as sh or bash.

ksh

Korn shell

derived from the Bourne shell so has a shared syntax.

job control taken from the C shell.

bash

Bourne-Again shell

Combines the "best" of sh, ksh, and csh.

Default shell (out of the box) on Linux and Mac OSX operating systems.

Pro: Flexible and powerful scripting language with all the interactive features of csh plus command completion.

This shell is great for complicated GMT scripts.

tcsh

TENEX C shell

Default shell of the CERI unix environment.

Pro: User friendly on the command line.

Con: It is not as suitable for long and involved scripts.

It is perfectly OK for most daily geophysics work on the command line & most faculty here use it on a daily basis so there are many experts around.

Features bash and tcsh Shells

Basics of the Unix/Linux Environment

Useful features of tcsh & bash

-file completion-

key the tab key, or the escape key twice, to “complete” the name of a long file.

Say I have a file named
`largest-deadliest-eqs-last-100-years.ai`

I can type just enough so the system can continue (i.e. there are no options for the next letter – assume I also have a file `lapilona.dat`)

```
$ls lar<tab>  will produce this
$ls largest-deadliest-eqs-last-100-years.ai
```

Useful features of tcsh & bash

-file completion-

Say I have 2 files file named

```
ls largest-deadliest-eqs-last-50-years.ai
ls largest-deadliest-eqs-last-100-years.ai
```

Actually I can type just enough so it can continue on its own for a while

`$ls lar<tab>` will produce this

```
$ls largest-deadliest-eqs-last-
```

At which point it gets stuck. I help it along

```
$ls largest-deadliest-eqs-last-1<tab>
$ls largest-deadliest-eqs-last-100-years.ai
```

Useful features of tcsh & bash

history command

list the previous commands entered during the active session.

```
148:> history
. . .
   145   21:30    pwd
   146   21:30    DEM
   147   21:30    cd srtm
   148   21:30    history
```

Useful features of tcsh & bash

-history “feature”-

Shell keeps “history” of commands

up and down arrow keys: allow you to move up and down through previous commands.

right and left arrow keys: allow you to edit command lines (backspace to remove, type at cursor to insert) without starting from scratch.

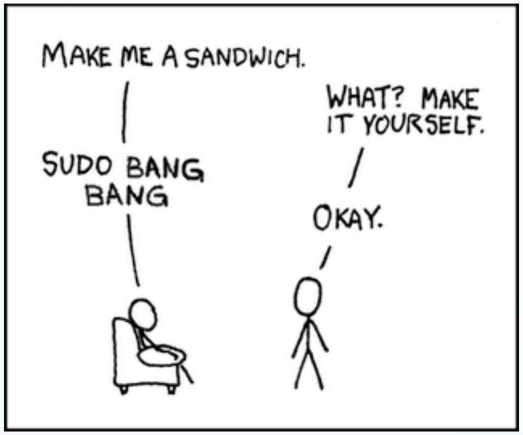
Useful features of tcsh & bash

bang (“ ! ”) command/shortcut

Bang is used to search backward through your *Bash/tcsh* history until it finds a command that matches the string that follows the bang and returns/executes it.

bang ("!") command/shortcut

`!!`: reruns the last command in the history list.



```
% vi foo.c bar.c
% !!
```

Becomes: `% vi foo.c bar.c`

`!vi`: reruns the last command in the history file beginning with "`vi`".

```
% vi foo.c bar.c
% ls
% !vi
```

Becomes: `% vi foo.c bar.c`

bang ("!") command/shortcut

!XXX<CR> returns the command numbered XXX in the history list. It runs it after you enter the <CR>.)

```
148:> history
. . .
   145   21:30    pwd
   146   21:30    DEM
   147   21:30    cd srtm
   148   21:30    history
149:> !146
DEM
/gaia/home/rsmalley/dem
150:>
```

bang ("!") command

!-X: returns the command X back in the history list and runs it at the <CR>.

```
151:> history
. . .
   147   21:30    cd srtm
   148   21:30    cd ~
   149   21:30    history
   150   21:46    DEM
   151   21:55    history
152:> !-4
cd ~
/gaia/home/rsmalley
153:>
```

bang (“!”) command/shortcut is actually more general – use it to return commands from history and do something with them.

For the purposes of these tips, every tip will assume these are the last three commands you ran:

```
% which firefox
% make
% ./foo -f foo.conf
% vi foo.c bar.c
```

Getting stuff from the last command:

Get the last argument (“$”) from command :

```
% svn ci !$
```

Becomes:

```
% svn ci bar.c
```

Various shells have options that can affect this.

Be careful with shells that let you share history among "instances" (if you have 5 terminals open you have a shell running in each one. Each running copy is an "instance"). You can also have shells running in the "background" (almost never needed with modern gui's, was essential with single terminal).

Some shells also allow bang commands to be expanded with tabs or expanded and reloaded on the command line for further editing when you press return.

bang ("!") command/shortcut.

For the purposes of these tips, every tip will assume these are the last three commands you ran:

```
% which firefox
% make
% ./foo -f foo.conf
% vi foo.c bar.c
```

Getting stuff from the last command:

All arguments ("*", special definition):

```
% svn ci !*
```

Becomes:

```
% svn ci foo.c bar.c
```

bang (“**!**”) command/shortcut.

For the purposes of these tips, every tip will assume these are the last three commands you ran:

```
% which firefox
% make
% ./foo -f foo.conf
% vi foo.c bar.c
```

Getting arguments from the last command:

First argument (“**:N**”):

```
% svn ci !!:1
```

Becomes:

```
% svn ci foo.c
```

bang (“!”) command/shortcut

For the purposes of these tips, every tip will assume these are the last three commands you ran:

```
% which firefox
% make
% ./foo -f foo.conf
% vi foo.c bar.c
```

Accessing command lines by pattern: (saw this already, but now with ./, need to go to first letter)

Full line:

```
% !./f
```

Becomes:

```
% ./foo -f foo.conf
```

bang (“!”) command/shortcut

```
% ls -d a*.f
atantest.f
% make
% ./foo -f foo.conf
% vi foo.c bar.c
```

Accessing command lines by pattern and command substitution:

This:

```
% vi `!ls`
```

Becomes:

```
% vi `ls -d a*.f`
```

Which becomes:

```
% vi atantest.f
```

bang ("!") command/shortcut

For the purposes of these tips, every tip will assume these are the last three commands you ran:

```
% which firefox
% make
% ./foo -f foo.conf
% vi foo.c bar.c
```

Accessing command lines by pattern:

All args : `% ./bar !./f:*`

Becomes: `% ./bar -f foo.conf`

We are looking for the command that begins with "./f", and then we want (the colon, ":") all of its arguments (the splat, "*")

bang ("!") command/shortcut

Notice how this makes perfect sense under the Unix philosophy.

Make a tool and (mis/ab)use it.

(the basic commands are really very simple, but in tricky combination they become very powerful - and confusing.)

Most normal people are not going to use all these shortcuts, they are just too complicated.

I showed them, however, to present additional application of the Unix philosophy.

When you Google for help with Unix the answers/ examples are usually maximally Unixified, so you will have to figure it out.

bang (“!”) command/shortcut

you can also view the command that bang finds without immediately executing it.

```
!cat:p<CR>
```

Now, instead of executing the command it finds, bang prints the command to Standard OUT for you to look at.

bang (“!”) command/shortcut

```
!cat:p<CR>
```

That's not all though, it also copies the command to the end of your history (even though it was not executed).

This is useful because if you do want to execute that command, you can now use the *bang bang* shortcut to run it (*bang bang* runs the last thing in history).

How typically Unix.

bang (“!”) command/shortcut

```
$ !cat:p<CR>
cat tst.sh
$ !! | grep "hello”<CR>
```

Here, the most recent command containing *cat* is printed, and copied to the end of your history.

Then, that command is executed with its results being piped into the *grep* command, which has been specified to print those lines containing the string "hello”.
(We are following Unix philosophy)

bang ("!") command/shortcut

To find a lot of this "neat" stuff, I GOOGLEd

"unix bang command"

---

you will not find it in the man pages

```
147:> man !
No manual entry for !.
148:>
```