A
Northern Illinois University
Academic Computing Services
Workshop

UNIX Basics for the Mere-Mortal User

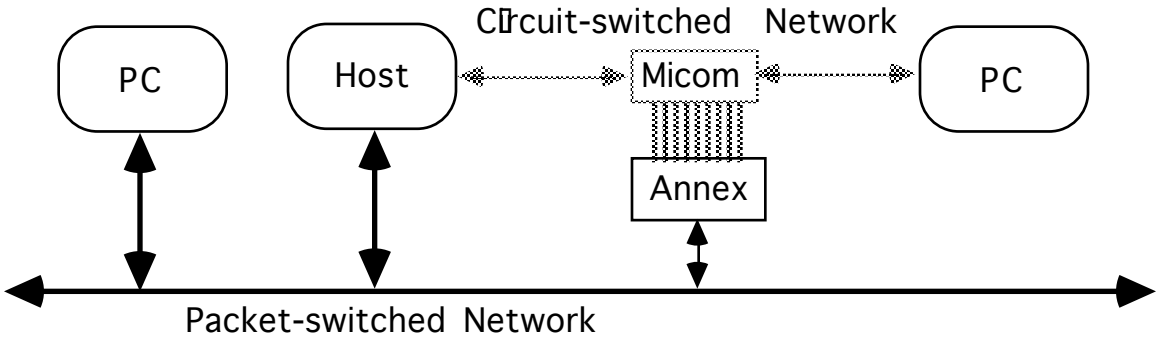
Michael G. Prais
Swen Parson 120
753-1057

This workshop is a series of examples that provide experience with basic UNIX operations. It expects that you have an account and physical access to a station on a UNIX system. The following topics are covered.

System Access
Process Management
File Management
Text Management
Text Processing
Print Management
Program Development
Termination

UNIX is a common operating environment for workstations, minicomputers, and supercomputers. UNIX systems are multiuser systems that act as hosts for several users to simultaneously enter commands and receive responses. UNIX is available in several variants (BSD, SysV, and others), but most of the commands are universal. Any differences in the commands described in this workshop are illustrated by giving the syntax of both commands. Most differences appear in command options, program development, and in system administration.

UNIX systems on campus are generally on the NIU packet-switched network "NIUnet". They can be reached by other workstations on this network or by other workstations directly connected to or dialing-in to the Micom RS232C circuit switch.



Network Access to a UNIX System

A UNIX System on the NIU packet-switched network can be accessed as a remote system from any personal computer that is also on the network and that uses the TCP/IP suite of communications software.
ACS has a Sun SPARCstation that is on the network.
The PCs in SP10A use a variant of the TCP/IP *telnet* command which provides remote access as a DEC vt220 terminal on the network.

tnvt220 nirvana — Accesses the ACS Sun SPARCStation.

The hostname *nirvana* is translated through a table on the PCs to the network of the ACS host system.

Micom Access to a UNIX System

The UNIX systems on the packet-switched network can also be reached by first going through the Micom circuit switch to get to the Annex terminal switch (*umax*) which is on the packet-switched network.
The Annex allows terminals and PCs acting as terminals on the circuit-switched network access to the packet-switched network.

The following steps describe how to reach the ACS Sun SPARCstation through the Micom from the Stevens Lab.

BREAK BREAK BREAK ENTER — Requests the Micom menu over a Data-Over-Voice (DOV) line. Another procedure is required for dial-in.

umax	Requests an Annex network connection.
ENTER ENTER	Requests the *annex:* prompt.
telnet nirvana.acs	Accesses the ACS Sun SPARCStation.

Because ACS is on a different subnet than the Annex,
you must use the hostname and subnet of the ACS host system.

Most UNIX systems present a *login:* prompt to check account access.
Enter your account username and press Enter to identify yourself.
A *password:* prompt is displayed.
Enter the account password and press Enter to verify your identity.
The password is not displayed as a security measure,
but if you know that you typed it wrong,
you can use BACKSPACE to erase erroneous characters,
and then retype the correct characters.
Successfully accessing a system through the *login:* and *password:* prompts
is often called *logging in*.

If the login/password combination does not match with the system values,
UNIX will respond with *login incorrect*, and redisplay the *login:* prompt.
Some systems may redisplay the *login:* prompt a limited number of times.

When the login/password combination is recognized by the system,
it displays several messages and finally a command line prompt.
If the system prompts for a terminal type, enter *vt220* and press Enter.

ENTER	Scrolls the screen and displays another prompt.
logout ENTER	Terminates your session.

Follow the previous instructions and re-access the system.

Process Management

A UNIX system is a multitasking system whose tasks are called *processes*.
The processes are generally started by commands typed or selected at a terminal interface. Typing a command after the % prompt is characteristic of the *C shell* command interpreter. Alternatives to the *C shell* include the *Borne* and *Korn (k) shells* which interpret typed commands and the *SunView*, *Motif*, and *Open Windows* graphical, point-and-click interfaces which interpret the motion of a pointing device such as a mouse or trackball.

The *C shell* (the command interpreter) uses any non-zero number of spaces or tabs to separate the parts of a command line.
The first word of the command line is the *command*
and the remaining words on the line are its *arguments*.
Pressing ENTER signals the shell to act on the command.

ps ENTER	Displays all processes connected with your terminal.
echo Hello ENTER	Commands the system to echo the argument *Hello*.

The symbol _ is used to identify a space that may too easily be missed.
The indication to press the ENTER key is omitted hereafter
unless it may too easily be missed.

Some commands are interactive and provide prompts.
The *man* command provides pages of helpful information.

man intro	Displays one page of the commands described in the on-line manual.
ENTER	Displays the next line of the description.
Space	Displays the next page of the description.
q	Quits the *man* listing.

The action of UNIX commands on their arguments are tailored by *options*. Options precede the arguments and are themselves often preceded by a minus or a plus sign.

man -k users	Displays a list of topics in the system manual that deal with *users*.
man leave	Displays one page of an on-line manual for the *leave* command.
leave	Commands the system to prompt *When would you like to leave?*
+300	Sets an alarm for three hours from now.

Information about the status of the processes currently active on the system is available through the *ps* command.

ps	Displays all processes started by you and connected with your terminal.

The processes are identified by their *process identifier* in the PID column and their *controlling terminal* in the TT column.
The *csh* command in the listing is your *C shell* command interpreter and the *ps* command is what you just ran. (The command sees itself.)
The *leave* command places itself in the background and disconnects itself from your terminal when it starts so it does not show up on the simple process list.

ps -x	Displays all processes started by you and *not* connected with a terminal.

Note the process identifier of the *leave* command.

Processes can be controlled by the *kill* command.
Do not kill your first *csh* command unless you want to disconnect yourself.

kill *PID*	Terminates an identified process.

You would normally kill processes that you have put in the *background* (as is described later), but you can also use it from another terminal to eliminate a runaway process at this terminal and return you to the *C shell* prompt.
The *leave* command requires a special option on the kill command.

kill -9 *PID*	Terminates without prejudice a stubborn, identified process.
ps -g	Displays all processes on the system.

The system supports simultaneous access by multiple terminals and users. The diversity of access requires that each user configure their terminal (tty) for acceptable operation.

stty all stty -a	Displays the terminal (tty) line settings. (BSD/SysV)

Notice the Control-key combinations that can be used at this terminal.

	Erase last character
	Kill last line
	Erase last word
	Reprint screen
	Flush pending output
	Treat next character literally
	Suspend current process
	Interrupt current process
	Quit current process with dump
	Stop terminal output
	Restart terminal output
	Signal end of file

stty erase ^h kill ^u	Sets the character erase to CTRL H and the line erase to CTRL U. Remember to separate *erase* and *^h* and *kill* and *^u* with spaces.

The key combinations CTRL H and CTRL U are accepted in place of the sequences ^h and ^u unless they are already set in which case they act to erase necessary characters.

stty all stty -a	Displays the terminal (tty) line settings. (BSD/SysV)

Each user on a UNIX system has a certain operating environment. This environment is characterized by information about the following components.

System
Controlling Terminal
User ID
Group ID
Environment Variables for Substitution
Command Interpreter
Home Directory
Current Directories of Files
Directories of Executable Commands

hostname	Displays the system name.
uptime	Displays system status.
tty	Displays the terminal being used to access the system.
id	Displays your account name. (Not on Encore.)
who am i	Also displays your account name.
users	Displays the other users on the system.
who	Also displays the other users on the system.
w	Displays the other users with their activities. JCPU indicates all user processes. PCPU indicates active user processes.

finger *your_username*	Displays a description of your account.
finger	Displays a description of the active accounts.
groups	Displays the work groups that include you. (BSD only)
printenv	Displays the variables in the prototypical environment that each process inherits as it begins. TERM USER HOME PATH SHELL
set	Displays the parameters available for the current process--the *C Shell*. argv[] cwd home path prompt shell status term user
set notify	Sets the *C shell* to announce the completion of background processes.
set	Displays the changed parameters.
unset notify	Sets the *C shell* to disregard the completion of background processes.
set	Displays the changed parameters.

set ignoreeof	Prevents an extra or inadvertent Ctrl d from signaling the interactive *C shell* that you are done inputting text (at your end of file) and want to logout.
set	Displays the changed parameters.
set history=20	Sets the *C shell* to save the last 20 entered commands.
set	Displays the changed parameters.
echo $TERM	Displays the value of a single variable.
set prompt=$USER\ \[!\]:\ set prompt=$LOGNAME\ \[!\]:\	Sets the command line prompt to include your account id, and the command line number.

The backslash (\) is used to escape the next character from interpretation by the shell (the command interpreter).
In this case the spaces would be interpreted as argument separators rather than as simple characters that should appear in the prompt.
Double quotes ("...") can be used to protect the spaces in a string of characters, but allow the shell to recognize variables and other features.
Single quotes ('...') protect all characters from interpretation by the shell. This type of quote provides complete literal interpretation.

The *C shell* remembers the commands that you recently entered.

history	Displays the more recent commands with their line numbers.
!!	Executes the most recent command, that is, *history*.
!1	Executes command number 1.
!ec	Executes the most recent command that begins with *ec*, that is, *echo $TERM*.

!?y	Executes the most recent command that contains the character *y*, that is, *history*.
^his^her	Executes the most recent command with the characters *his* replaced with *her*, that is, *hertory*.

```
                                    hertory:  Command not found.
```

This response is typical of a nonexistent (perhaps misspelled) command.

The environment must be set to the appropriate terminal type for it to function properly in full screen mode for programs such as the editor.

tset -r	Displays the terminal type of your access device.
set noglob	Ignores filename wildcards.
eval `tset -sr vt220`	Sets the terminal type for your session to a DEC vt220.

The ` ... ` construction is used in UNIX to execute the contained instructions prior to the rest of the command line,
and substitute the resulting output into that command line
for the final execution.
The *-s* option of *tset* produces a string of commands
to set the terminal type and *eval* executes those commands.
This set of commands includes an *unset noglob*.

set noglob	Ignores filename wildcards.
tset -sr vt220	Displays the commands generated to set the terminal type.
stty all	Provides another look at the terminal control-keys.

yes yes	Presents a continuous positive response.
CTRL C	Interrupts and terminates this foreground process.
yes no	Presents a continuous negative response.
CTRL Z	Suspends the current foreground process.
jobs	Displays the background processes that are Running or Stopped.
bc	Starts up an interactive calculator process.
scale=4	Sets the output precision to 4 digits to the right of the decimal.
1+1	Exhibits a well-known mathematical result.
CTRL Z	Suspends the current interactive process.
jobs	Displays the active processes.
kill %1	Terminates process 1, that is, *yes no*.
jobs	Displays the active processes.
fg %2	Resumes process 2, that is, *bc*.
CTRL D	Signals end of input to and terminates the interactive process.
date	Displays the date and time as an example of a simple command.
date ; date	Illustrates that multiple commands can appear on a single line (before pressing ENTER) by separating them with a semicolon.

date ; \ENTER date	Illustrates (with a short example) that extended commands can be placed on subsequent lines by escaping the ENTER with a backslash just before it.
sleep 5	Exhibits a *foreground* process that does not return a prompt immediately.
sleep 20 &ENTER echo Hey\! Wake up.	Exhibits a process that is placed in the *background* so that the command line prompt is redisplayed and a second process is started. The backslash is necessary to quote the !.
sleep 60 &	Starts a longer background process. Note the [job] and process numbers displayed.
ps	Displays all processes connected with your terminal. Note the process identifier (PID) of the most recent background process.
kill *pid*	Terminates the process with *pid*.
sleep 60 &	Starts another background process.
jobs	Displays the background processes.
kill %\?ee	Terminates the background process that contains an *ee*. Without the backslash, the question mark is considered a wildcard by the *C shell*.

File Management

UNIX uses a hierarchical file structure with files organized in directories.
Each file and directory (itself a file) carries a set of attributes that include the name of a user and a group that can access the file and a set of permission that describe how a user, a group, and others can use the file.
Files are identified with a path name with respect to the *root* of the file system (an absolute path name) or with respect to the current directory (a relative path name). Absolute path names start with a slash (/) and directories in the path to the file are separated with slashes.

/home/mgprais/mydir/myfile
/usr2/mgprais/mydir/myfile

Relative pathnames do not start with a slash, but subdirectories of the current directory on the path to the file are separated with slashes.

mydir/myfile

set noclobber	Sets the *C shell* to prevent blithely overwriting exiting files.
pwd	Displays (prints) the current working directory.
ls	Displays a list of the files in the current directory.
touch myfile1	Creates an empty new file.
ls	Displays a list of the files in the current directory. Note the presence of *myfile1*.
cat > myfile2 Line 1 Line 2 Line 3 Ctrl d	Creates (if necessary) and fills a new file. Signals the end of input to and terminates the *cat > myfile2* command.

The command *cat* is an abbreviation for *catenate - to connect*.
It can be used to connect and display a series of files,
but given only a single file, it displays the contents of that file.
Without arguments *cat* displays what is typed at the terminal.
The symbol > redirects the output of *cat* (or any command) to a file.

cat myfile2	Displays the contents of a file.
cat >> myfile2 Line 4 Line 5 CTRL D	Appends text to an existing file. Signals the end of input to and terminates the *cat >> myfile2* command.
cat myfile2	Displays the contents of a file.
cat > myfile2	Attempts to replace an existing file with new text while *set noclobber* is on. myfile2: file exists.
cat >! myfile2 Line 1 Line 2 Line 3 Line 4 CTRL D	Replaces an existing file with new text while *set noclobber* is on.
cat myfile2	Displays the contents of a file.
mkdir mydir	Creates a subdirectory.
ls	Displays the new subdirectory among the existing files
cd mydir	Changes the current working directory.
pwd	Displays the current working directory.

cd	Changes the current working directory to your home directory. (This is unlike the MS DOS *cd* command.)
pwd	Displays the current working directory.
cp myfile2 myfile3	Makes a duplicate of a file with a new name in the same directory.
ls	Displays the new file among the existing files.
cp myfile2 mydir	Makes a duplicate of a file with the same name in a new directory.
ls mydir	Displays a list of files in a subdirectory.
cp /home/michael/workshops/dante **.** cp /usr2/michael/workshops/dante **.**	Makes a duplicate of a file from the directory of *michael* in the current directory. Note the use of an absolute pathname.

The current directory is identified for quick reference by a dot (.).

mv myfile2 myfile4	Renames a file.
ls	Displays the list of files in the current directory. Note that *myfile2* is absent and that *myfile4* is present.
mv myfile4 mydir	Moves a file into another directory.
ls	Displays the list of files in the current directory. Note that *myfile4* is absent.

ls mydir	Displays the list of files in a subdirectory. Note the presence of *myfile4*.
ln myfile3 myfile5	Gives the first file a second name.
ls	Displays the list of files in the current directory.
ln myfile5 mydir	Gives a file in the current directory a name in the directory *mydir*.
ls mydir	Displays the list of files in a subdirectory. Note the presence of *myfile5*.
ln mydir/myfile2 **.**	Gives a file in the directory *mydir* a name is the current directory. Note the use of a relative pathname.
ls	Displays the list of files in the current directory.

The abbreviation *~user/* can be used as a replacement for the pathname of the home directory of user. The abbreviation *~/* can be used as a replacement for the pathname of your home directory. These pathnames may be long, changeable, or perhaps too much trouble to keep track of.

echo ~	Displays your home directory.
echo ~michael	Displays the home directory of user *michael*.
ln ~michael/workshops/program.c **.**	Gives a file from the home directory of *michael* a name in the current directory.
ls	Displays the list of files in the current directory.

Changes to a linked file affect a single, common copy of the file which can be very powerful if planned and very dangerous if not.

rm myfile2	Removes a file(s) from the current directory. (*rm* is set to prompt for your confirmation.)
y	Confirms the removal.
ls	Displays the list of files in the current directory. Note the absence of *myfile2*.
rm -i myfile3	Removes a file from the current directory after a positive response to a prompt.
n	Avoids the removal of the file(s).
rm mydir	Does not remove a directory.
rmdir mydir	Does not remove a non-empty directory.
rm -r mydir	Removes a non-empty directory.
ls	Displays the list of files in the current directory. Note the absence of *mydir*.
ls -lg ls -l	Displays the permissions and ownership of the files in the current directory. (BSD/SysV) Type (directory, symbolic link, regular, ...) Mode (permissions) Links Owner Group Size in characters Last Modified Name

Notice the number of different places where *program.c* shows up (links).

The long listing of the files in a directory displays the *mode* of the files as a string of nine characters *rwxrwxrwx* which indicates read, write, and execute permissions for the owner (user), group members, and others. A dash (-) in place of any character indicates the absence of permission.

Write permission for files allows changing the contents of the file, but does not allow creation or deletion of the file.
Write permission for the directory that contains the file allows creation and deletion of files in that directory.
Execute permission for files allows them to be interpreted as instructions.
The equivalent permission for directories allows the directory name to be used in (to be used to extend) a pathname.
Without this ability files within the directory and its subdirectories can not be executed.

ls -lgd ls -ld	Displays the permissions and ownership of the current directory itself. (BSD/SysV)
chmod go-r myfile3	Removes read permissions for group members and others.
ls -lg	Displays the permissions and ownership of the files in the current directory. Note the changes for *myfile3* and for *myfile5*, its othername.
chmod u-w myfile3	Removes the ability for the user to change (write to) a file.
cat >> myfile3	Attempts to append text to the protected file.
chmod u+w myfile3	Allows the user to change a file.
cat >> myfile3	Append text to the writable file.

Line 5
CTRL D

chmod u-w **.**	Removes the write permission from the current directory for the user.
ls -lgd	Displays the permissions and ownership for the current directory.
rm myfile3	Attempts to remove a protected file.
chmod u+w **.**	Provides the user (owner) of the current directory with create and delete permissions.
rm myfile3	Removes a file from the current directory with create and delete permissions. Notice that *myfile5* remains.

The *C shell* provides a way to replace long and/or frequently used commands with simple abbreviations.
Any spaces in the command to be replaced must be escaped or the whole command must be quoted to prevent the *C Shell* from recognizing the space as the end of a word or argument.

alias rm "rm -i"	Allows commands to be redefined.
rm myfile1	Prompts for a positive response before removing a file.
n	Avoids removing the file.
\rm myfile1	Avoids the *rm -i* alias and removes the file without prompting.
unalias rm	Removes the alias to *rm*.
alias	Displays the current aliases.

alias dir ls alias type cat alias copy cp alias rename mv alias ren mv alias md mkdir alias rd rmdir alias del "rm -i" alias erase rm	Defines an MS DOS environment.
alias	Displays all aliases.
copy myfile5 myfile6	Makes a duplicate of a file under a new name.
md mydir	Creates a new subdirectory.
dir	Displays a list of files and subdirectories in the current directory.
copy myfile5 mydir	Makes a duplicate of a file in a subdirectory.
dir mydir	Displays a list of files and subdirectories in a subdirectory.

Before proceeding be sure that the following files and directories exist: *dante*, *mydir*, *mydir/myfile5*, and *program.c*.

Text Management

The contents of files can also be viewed and manipulated.

more dante	Displays the first screen of a file and waits for a command.
ENTER	Scrolls the next line in the file onto the bottom of the screen.
SPACE	Displays the next screen of the file.
? h	Displays a list of the commands for *more*. (BSD/SysV)
q	Terminates *more*.
more +4 dante	Displays a file on screen with line *4* at the top of the screen with one line above it and waits for further instructions.
=	Displays your location within the file.
q	Terminates *more*.
more +/Four dante	Displays a file on screen with a line containing *Four* at the top of the screen and waits for further instructions.
q	Terminates *more*.
vi +/Seven dante	Displays a file for full screen editing with a line containing *Seven* at the middle of the screen.
:q	Exits the editor without changing anything.

vi +7 dante	Displays a file for full screen editing with line *7* at the middle of the screen.
CTRL G	Displays the file name and the current line number.
:q	Exits the editor without changing anything.
vi + dante	Displays a file for full screen editing with the last line of the file at the middle of the screen.
:q	Exits the editor without changing anything.
vi	Opens a screen that can be used to create a new file.

The visible editor *vi* starts up in a mode in which everything typed is interpreted as an editing command.

CTRL G	Displays the file name and the current line number.
o	Opens the editor in a mode in which almost everything typed is inserted as text.
Line 1 Line 2	Enter some text.
ESCAPE	Returns the editor to command mode.
:q	Exits the editor only when changes were not made.
:q!	Exits the editor discarding any changes that were made.

vi dante	Displays a file for full screen editing with the first line of the file at the top of the screen.

The visible editor *vi* reads a file into an internal buffer
and changes the buffer (not the original file) on command.
The original file is only replaced on command.

Vi has three modes:
the command mode, the insert mode, and the line mode.

:set showmode	Displays a note at the bottom right of the screen whenever in an insert mode.

The *command mode* is the central mode
that gives access to the other modes
and allows movement through the displayed file.
Commands in *vi* are case sensitive, so make sure the Caps Lock is off.

ESCAPE	Causes the system to beep when you are in the command mode.

Since the system beeped, you are now in the command mode.
If the system did not beep, you *were* not in the command mode
when you pressed Escape,
but since you did press Escape, you *are* now in command mode.
The next time you press Escape , the system will beep.

CTRL G	Displays the file name and current line number.
ENTER	Moves the cursor to the start of the next line.
RIGHT	Moves the cursor right to the next character.
}	Moves the cursor to the next blank line (the end of the current paragraph).

Most *vi* commands take a preceding multiplier.

5RIGHT	Moves the cursor five characters to the right.
5ENTER	Move the cursor to the start of the fifth line below.
>>	Indents the current line.
<<	Unindents the current line.
5>>	Indents the current and next four lines.
J	Joins the current and next line.
u	Undoes the last change to the buffer.
~	Changes the capitalization of the current character.
.	Repeats the last command.
U	Undoes changes to the current line. Changing lines looses the changes made in the previous "current" line.
W~	Capitalizes the next word.
W~	Capitalizes the next word.
mm	Marks the character as position m as one of 26 (a - z) points of reference.

Cursor Movement Commands

1G First line in file

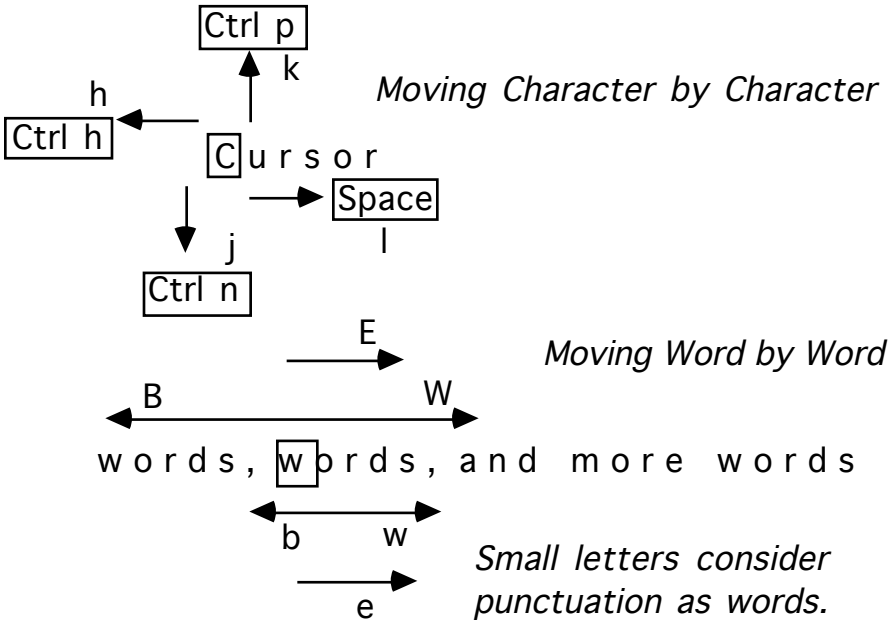
'm Line marked with position m

`m Character marked as position m

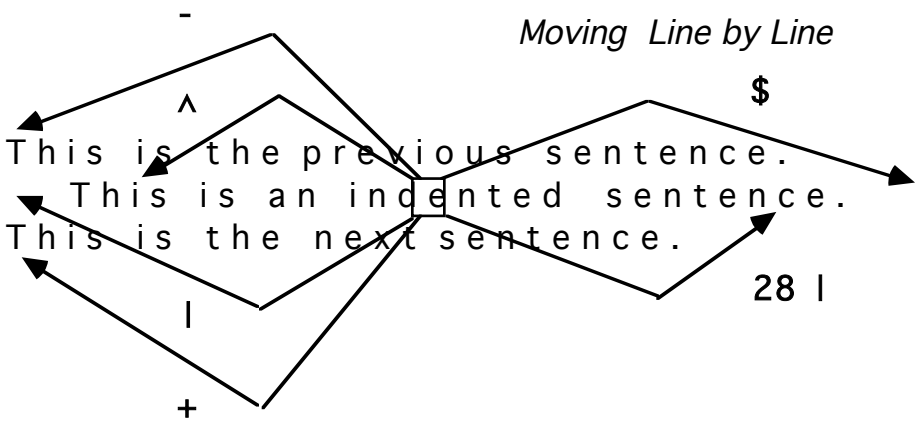
'' *Returns the cursor to its previous line*

`` *Returns the cursor to its previous character*

H Top of screen



M Middle of screen



L Last line on screen

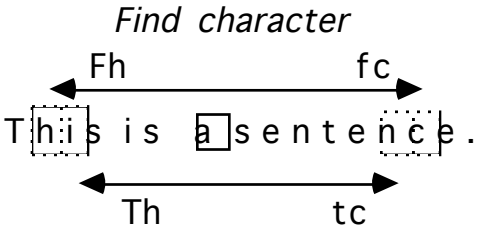
nG Line n in file

G Last line in file

Window Movement Commands

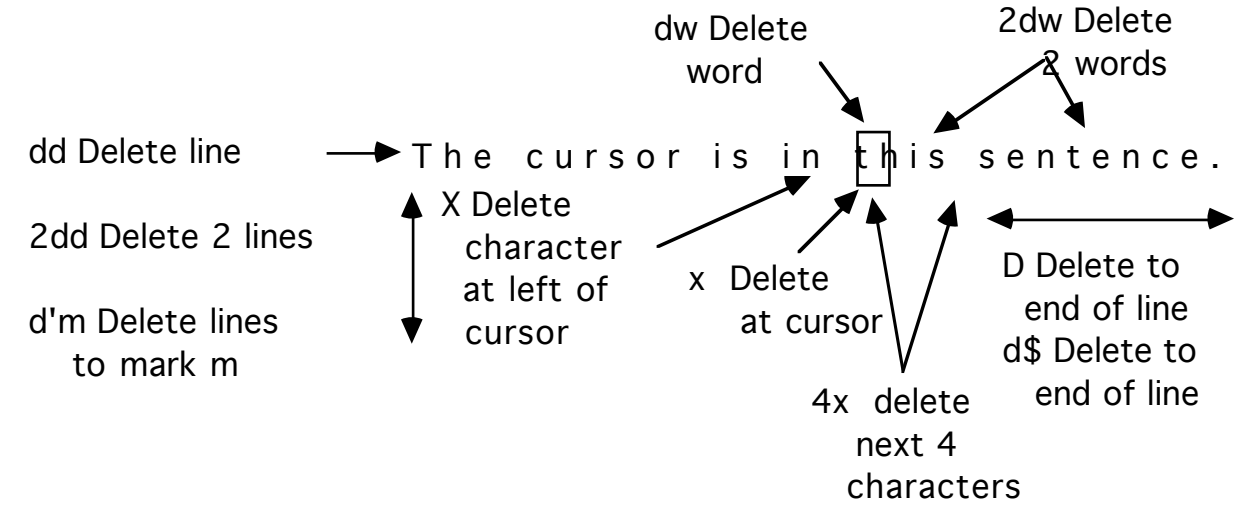
Scroll Screen:		Place Current Line:	
CTRL B	Up/Back one screen	z+	At top of screen
CTRL U	Up/Back half screen		
CTRL E	Up/Back one line		
---- Current Line ----		z.	At middle of screen
CTRL Y	Forward/Down one line		
CTRL D	Forward/Down half screen		
CTRL F	Forward/Down one screen	z-	At bottom of screen

Find in Current Line:

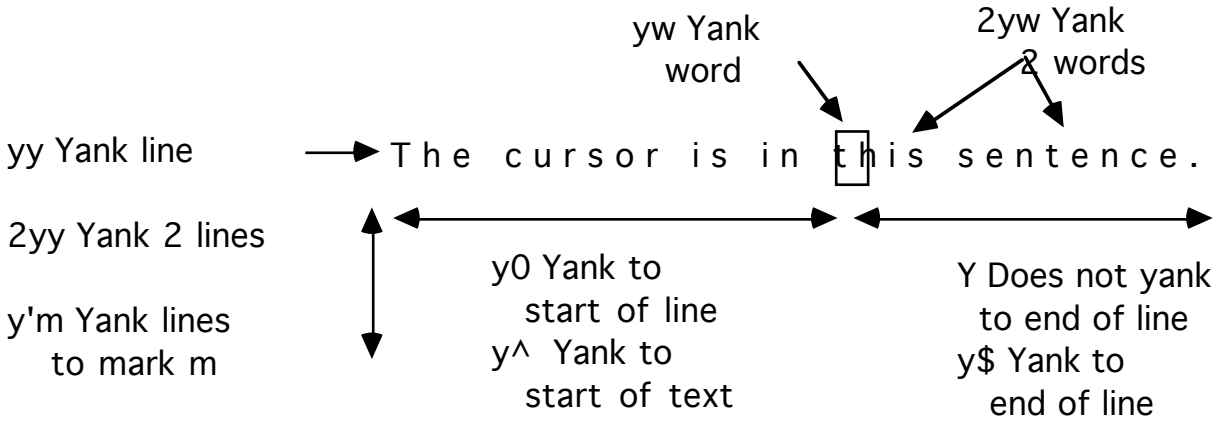


; Repeat Last Find

Delete (Cut) Text



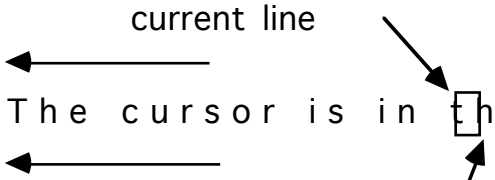
Yank (Copy) Text



Put (Paste) Text

P Put characters before cursor or put lines before current line

Text that is deleted or yanked is placed in a series of buffers for later use.



p Put characters after cursor or put lines after current line

Deletes, *yanks*, and *changes* place text in an unnamed buffer
which is used by the *put* commands.
There are also 26 named buffers (a through z) in which to hold text
during *deletes* and *yanks*.

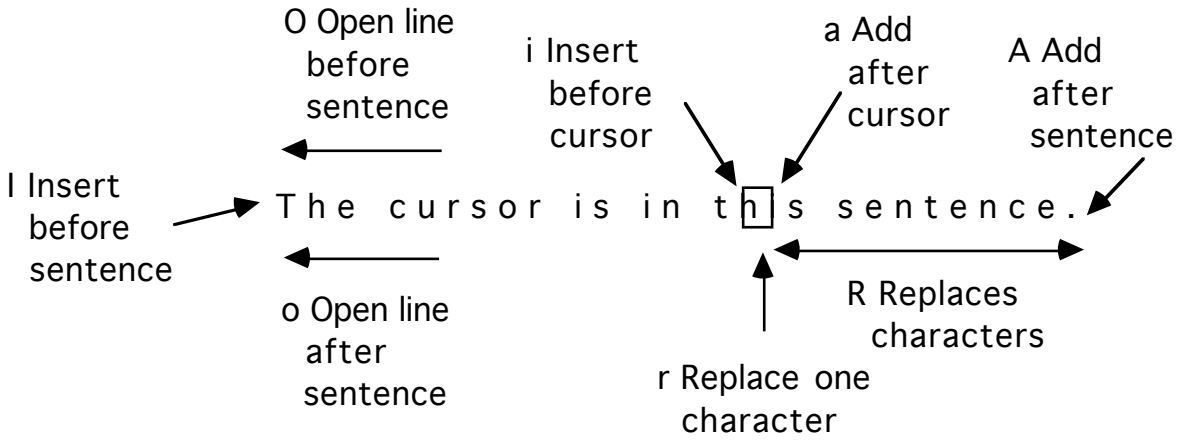
"adw	Deletes word into buffer *a*.
"ap	Puts the contents of buffer *a* after cursor.
"A5yy	Yanks and appends the next five lines to buffer *a*.
"ap	Puts contents of buffer *a* after cursor.

The buffers hold either words or lines; one cannot be appended to the other.

The *insert mode* allows text entry and replacement.
Any of the insert or change commands can be used to start a new file.

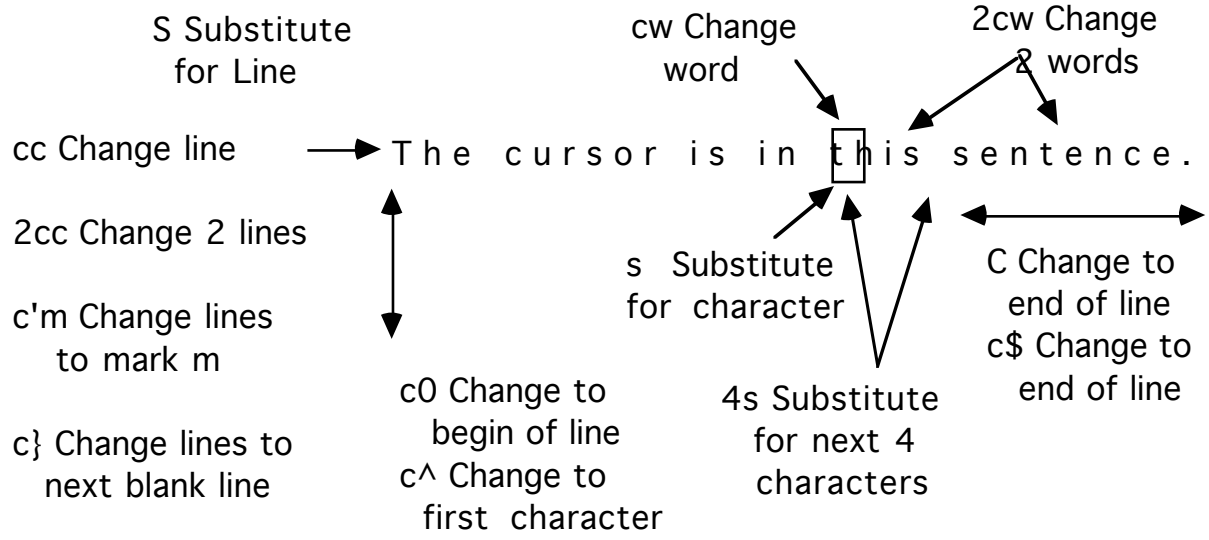
ESCAPE	Changes the system into command mode when you are in the insert mode.
BACKSPACE	Deletes the last character entered while in insert mode.
CTRL W	Deletes the last word entered (up to the last space) while in insert mode.
CTRL U	Deletes the last line entered.

Insert Text



Type text immediately after the command, and press Escape to return to Command mode.

Change Text



Notice the $, type text immediately after the command, and press Escape to return to Command Mode.

The text that was changed is saved in the delete buffer and can be placed elsewhere.

Line mode offers extensive search and replace capability and file manipulation. The symbols /, ?, :, !, and Q place the cursor at the bottom of the screen and place *vi* in line mode.

:	Places the cursor at the bottom of the screen and starts the line mode.
ESCAPE	Returns the system to command mode when you are in line mode and aborts any unexecuted commands.

Vi returns to command mode automatically after the line mode commands are completed unless Q was used to place *vi* in line mode.

Q	Places *vi* in line mode permanently.
vi	Returns from permanent line mode.
/Level	Searches for the first occurrence of *Level* toward the end of the file.
/	Searches for the next occurrence toward the end of the file.
?	Searches for the next occurrence toward the start of the file.
:s/is/IS/	Replaces the first occurrence of *is* in the current line with *IS.*
://s	Replaces the next occurrence.
:g/ is /s// IS /cg	Searches for all occurrences of the word *is* (surrounded by blanks), checks for confirmation (y), and replaces them with the word *IS* (surrounded by blanks).

:w	Writes out the buffer on to the original file.
:1,.w first_part	Writes out the first part of the buffer (up to and including the current line) and creates a new file.
:.,$w >> first_part	Appends the last part of the buffer (starting with the current line) to a file.
:e first_part	Begins editing another file. Note the duplicate line.
:e#	Returns to the original file.
:e!	Starts editing an original version of the file.
:f new_dante	Changes the name of the output file.
:cd mydir	Changes the current directory in which files are accessed.
:r myfile5	Reads in the file myfile5 after the current line.
:cd ..	Changes the current directory in which files are accessed.
:r!date	Reads the output of the command date into the buffer at the cursor.
:!ls	Displays the files in the current directory.
o	Opens a new line to insert text.
532+694	Enters a new line of text.
ESCAPE	Returns to command mode.
!!bc	Uses the current line as input to *bc*, and replaces the current line

	with the output of the command *bc*.
!}sort	Sorts the text to next blank line and replaces it with the sorted text.
:sh	Starts a new shell without quitting *vi*.
exit	Returns to the edited file.

A series of commands can yanked or deleted into a buffer
and then simply executed over and over again to avoid retyping.

o	Opens a new line to place text.
W~ESCAPE	Enters as text the command to capitalize the next word and returns to command mode.
lb	Moves back to the beginning of the command.
"ad$	Deletes the command text into buffer *a*. The end of line is not included.
@a	Executes contents of buffer *a* (as a macro).
@@	Executes last macro.

Keys can also be redefined to execute other commands.
The use of function keys is great for saving long substitutions across files.
These same commands can be placed in your *.exrc* file
and automatically executed when *vi* or *ex* start up.

:map Y y$	Identify the key Y as the command y$, that is, redefine Y as *yank* to the end of the line so that it is similar to the commands D and C.

:map #1 iandCTRL V ESCAPE	Identify the key F1 to insert *and* before the cursor. CTRL V is used to protect the ESCAPE key from immediate action.
:map	Displays the key mappings.
:x	Exits *vi* after saving all changes.
:q	Exits *vi* when no changes have been made.
:q!	Exits *vi* without saving changes.

Text Processing

wc dante	Displays the number of characters, words, and lines in a file.
head dante	Displays the first 10 lines of a file.
tail -15 dante	Displays the last 15 lines of a file.
grep "Two" dante	Displays the lines in a file containing the characters *Two*.
grep -v "Def" dante	Displays the lines in a file not containing the characters *Def*.
sort dante	Orders the line in a file.
comm -12 first_part dante	Compares two (sorted/similar) files and displays the lines found in both files.
sed -e "s/Level/Line/g" dante	Replaces text in a file line-by-line for each occurrence in the line.

The *sed* command puts each successive line of a file in a buffer
and then runs all instructions against it; what is left is printed.
Other useful instructions for changing text are *d*, *a*\, *i*\, and *c*\.

The *awk* command is useful for selecting and printing information
from columns within a file.

awk '/Three/{print $2, $1}' dante

For each line containing *Three*,
exchange the first and second words (columns)
separated by spaces or tabs.

These text processing commands are useful when they are used as *filters*
to change text and pass it to another command.
These commands can be "connected by pipes"
where the output to one command is directed into the input of another.

head dante | sort	Sorts the first 10 lines of a file.
cat -v -t dante	Displays a file with visible control characters except newlines and formfeeds. Tabs are listed as **^I**.
cat -v -t dante | more	Displays a file with visible control characters except newlines and formfeeds one screen at a time.
sed -e "s/ /\\ENTER /g" dante | sort -u > words	Separates and sorts all unique words in a file. Note that the ENTER was escaped by the double backslash.
cat words	Displays the file.

Other useful commands include *tr*, *cut*, *paste*, *join*, *uniq*, *look*, *diff*, *split*,
fmt, and *spell*.

Print Management

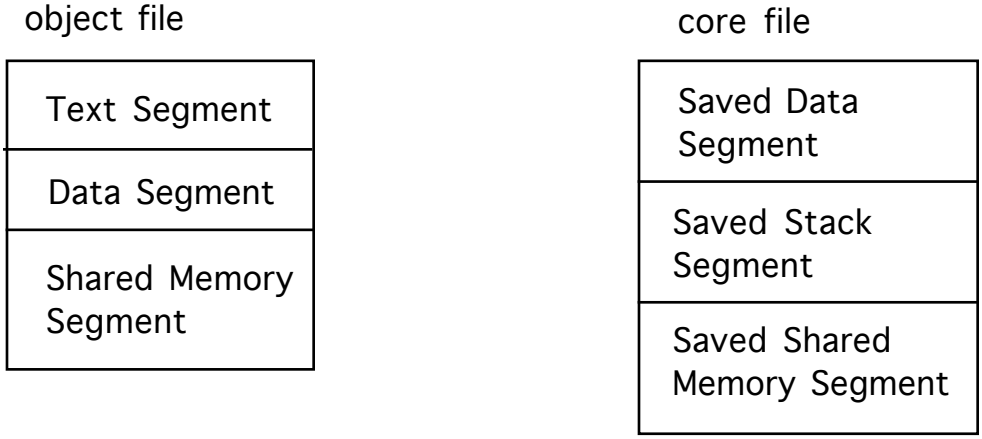
lpr myfile5	Sends a file to be printed.
pr myfile5 | more	Formats a file into pages with headers.
pr myfile5 | lpr	Formats a file into pages with headers and sends it to be printed.
pr -h "A Special Header" myfile5 | more	Formats a file into pages with a supplied header.
pr -t myfile5 | more	Formats a file into pages without headers
pr -m myfile5 myfile5 |more	Formats individual files into multiple individual columns on pages.
pr -3 words | more	Snakes the lines from a file onto pages with 3 columns.
lpq lpstat	Displays the files in the print queue. Note the numbers associated with each job. (BSD/SysV)
lprm *job#* cancel *job#*	Removes an identified print job from the queue. (BSD/SysV)

Program Development

cc program.c	Compiles C source code into object module *program.o* and loads them into an executable module *a.out*.
a.out	Executes the compiled C source code.
cc program.c -o program	Compiles C source code into object module *program.o* and loads them into an executable module *program*.
program	Executes the compiled C source code.
file program	Displays the file type of a file.
strings program	Displays the variable names from the symbol table of an executable module.
nm program	Display variable definitions from the symbol table of an executable module.
f77 program.f	Compiles Fortran source code into object module *program.o* and loads them into an executable module *program*.
ld c_program.o f_program.o	Loads C and Fortran object modules into an executable module *a.out*.
cc -g program.c	Compiles C source code for debugging into the object module *program.o* and loads them into an executable module *a.out*.
man -k debug	Displays debugging programs on the system.

dbx xdb sdb	Prepares to execute module *a.out* in the debugging environment.

The debugger examines an executable object module (*a.out* by default) and the image in memory (*core* by default).
The executable module as a file contains a header, the program instructions, the program data, reallocation information, a symbol table, and a string table.



The following commands are used in the *dbx* environment.

trace	Display activity during execution.
run	Starts executing *a.out*.
CTRL C	Interrupts the program.
help	Displays the available commands.
where	Describes the point of execution.
list	Displays ten lines of source code.
print position	Displays the value of the variable *position*.
step	Executes the next statement.

cont	Continues execution.
kill	Terminates execution.
quit	Exits the debugger.

The following commands are used in the *xdb* environment.

r	Starts executing *a.out*.
CTRL C	Interrupts the program.
h	Displays the available commands.
l	Displays information about the debugger.
L	Displays the statement being executed and its location.
v	Displays ten lines of source code.
s	Executes the next statement.
p position	Displays the value of the variable *position*.
M	Displays the address maps.
c	Continues execution.
k	Terminates execution.
q	Exits the debugger.

The following commands are used in the *sdb* environment.

1v	Toggles display of source code by step.
r	Starts executing *a.out*.
CTRL C	Interrupts the program.

Q	Displays a list of procedures and files being debugged.
l	Displays the current line.
w	Displays ten lines of source code.
s	Executes the next statement.
position	Displays the value of the variable *position*.
M	Displays the address maps.
c	Continues execution.
k	Terminates execution.
q	Exits the debugger.

time program	Displays elapsed, system, and user times for the execution of *program*.
set time=120	Displays elapsed, system, and user times for all executions of greater than 120 seconds.
limit cputime 120	Interrupts all commands once they have used more than 120 seconds cputime.

In developing a program, consider that you can suspend the editor by pressing CTRL Z, compiling and debugging your source code, and then return to the editor using the *fg* command.

Termination

The *ls* command does not normally list files that begin with a dot (.)
so that some seldom-used files can be selectively hidden from view.

ls -a	Displays all files including those with a dot (.) as the first character.

The environment used in this session can be saved for the next session by
editing two files that are run whenever a user logs in.
Both of these files are normally hidden from view
because they are seldom used once they are set up.
The commands in the file *.cshrc* are executed
whenever the user starts a *C shell* to interpret commands.
Place any *aliases* and *set* variables in your *.cshrc*.
The commands in the file *.login* are run after those in *.cshrc*
whenever the *C shell* is started from a login.
Place your *tset* command and any *setenv* variables in your *.login*.

vi .login	Edits the commands that set up your preferred environment.
source .login	Executes all the actions in *.login*. This can be used for testing.

The *tset* command in the *.login* file can be used to request a terminal type
if you access the system from several terminals.

grep 'l' /etc/termcap | more grep 'l' /etc/terminfo/? | more	Lists the abbreviations and names for all terminals known to the (BSD or SysV) system.

set noglob ; eval `tset -sr -m :?vt100`

Presents a terminal type
for your access device

TERM = (vt100)

and uses *vt100* when you press ENTER
although it accepts typed alternatives.

The *C shell* allows your background processes to continue
once you logout. An unnecessary process uses system resources.

sleep 600 &	Starts a useless process in the background.
ps	Displays the processes associated with your terminal.
login -p	Terminates your session and starts a new session with the *login:* prompt. The option -p preserves your environment. Use your username and password again.
ps	Displays the processes associated with your terminal.
kill *pid*	Terminates the process *sleep 600*.

Since the account that you used is not your own,
clean up any files your created.

rm -ir ~/*	Prompts you before removing all the files and subdirectories in your home directory.
logout	Terminates your session.

On your own account you can and should regularly change your password. Recommended passwords are words that are easy to remember and type, that do not appear in print anywhere with your name, that do not appear in a dictionary, that have upper and lowercase letters, and that have imbedded either punctuation marks or numerals.

passwd	Prompts for you to enter your old password, the word you want for your new password, and your new password again in order to change your password.