

## UNIX Information Handout

All of the work for this course is to be completed on the Suns in UNIX in C++ using the GNU C++ compiler. This handout is for your reference in learning to work in this environment.

### **Accessing the Suns**

Accessing to the Suns can be done in either of two ways – using Exceed on Demand, or using ssh. In either case, you must log in. If you had an account before, you still have the same account. Otherwise, your username is the same as your ULID, and your initial password is fall06.

There is a separate handout that covers accessing the Suns using Exceed on Demand.

To use the Suns from home, you must have some kind of internet connection. Once you are on the internet, you can use Exceed on Demand or you can use SSH (a secure version of telnet) to connect ironside.ilstu.edu. A good, free SSH client called putty is available on the T: drive in the 279 folder. Always connect to ironside.itk.ilstu.edu. Do not work on ironside. After you connect, you need to **ssh** to an alternative machine:

```
ironside% ssh pelleas
```

pelleas is the name of one of the machines that you would connect to. The machines available are listed when you first login in to ironside. When you do the second ssh, you may be asked for a password. Simply use your UNIX password.

## Command line vs. GUI

When using ExceedOnDemand, you will have access to a full GUI environment. You should feel free to explore this environment and use it. However, this handout focuses on how to do things in the command line environment, which is how you do things with ssh, and which is necessary for some purposes.

### Getting a command line window

If there is a terminal or console window on your screen, you can activate it by simply clicking in it. If no window is visible, right-click on the desktop and select *Hosts* and then *Terminal Console*. Notice that right-click on the desktop brings up a menu with a number of choices. You may want to explore this menu.

### Changing your password

One of the first things you should do the first time you log in to one of the Suns is change your password. Choose a password that you can remember, but one that is not a dictionary word. It is recommended that a password contains at least three of the following four possibilities: lowercase letters, uppercase letters, numerals, and punctuation. Passwords should be at least 6 characters long (8-12 characters is usually best).

To change your password, you should use the command **passwd**:

```
% passwd
Changing password for your_login_name
Old password: your_old_password
New password: your_new_password
Re-enter new password: your_new_password
```

### Leaving

To logout from an SSH session, you can simply type **logout**. With EoD, you'll find a little button at the bottom of the screen, slightly to the right, that says "EXIT". Click that button to logout when you are finished.

### Directory Structure

UNIX maintains a hierarchical file structure consisting of a root directory and several layers of subdirectories. These are much like the folders in the Windows and Macintosh operating systems. Each directory contains zero or more files and subdirectories. The root directory is always named / and referred to by this character.

At any point in time, each user is associated with a single directory called the *working (or current) directory*. Immediately after logging in to a UNIX system, a user is associated with a *home directory*. A possible home directory would be named **/home/tpaine**, where "tpaine" is the person's login name. Note that UNIX uses a slash to separate directory names, unlike Windows which uses a backslash.

## Naming Files and Directories

Almost any character can be used in the name of a file or directory, but try to use only the following characters: letters (upper or lower case), digits, underscore, hyphen, and period.

Either an absolute or a relative path name may be used whenever it is necessary to specify the name of a file or a directory.

To write an absolute path name, start with the root directory and list, in order, all of the directories that contain the desired object. Place the name of the desired object at the end of the list and separate all names from one another with the / character. For example, suppose that a file named **liberty** is in the home directory of user **tpaine**. The absolute path name of this file would be **/home/tpaine/liberty**. Remember never to use / to begin a relative pathname; you only use / to begin an absolute pathname.

A relative path name can be used to specify where a file is in relation to the working directory.

### Example 1

Suppose that a file named **barney** is in the working directory. The relative name for this file would be **barney**.

### Example 2

Suppose that a file named **dino** is in directory named **pets** which is immediately subordinate to the working directory. The relative name for this file would be **pets/dino**.

. and ..

The period and the double period are important symbols in dealing with directories. The single period refers to the working directory. This means that **./pets/dino** is equivalent to **pets/dino**. The double period refers to the parent of the working directory.

## Commands for getting around

### **pwd**

Stands for print working directory. Prints the absolute pathname for the current directory.

### **cd**

Stands for change directory. Use this command to switch directories. It typically takes one argument (the directory to move to). **With no argument, it makes the user's home directory the working directory.**

Examples:

% **cd pets**

Will change the working directory to pets, a subdirectory of the current working directory

**% cd ../program2**

Will change the working directory to program2, which must be a “sister” directory of the current working directory (i.e. the two directories have the same parent directory).

**% cd**

Will change the working directory to the user’s home directory—on our system typically */home/username*.

**ls**

Use this command to list the files in the working directory. You can give it a directory name to see the contents of a particular directory. The ls command has several useful options. The most used of these is *-l*, which gives a “long” file listing with more detailed information about each file.

**dir**

This is an alternative command for listing the files. It provides a long listing similar to the results of *ls -l*.

## Commands for dealing with directories

**mkdir**

Stands for make directory. Use to create a new directory within the working directory. Takes one argument: the name of the new directory.

**rmdir**

Stands for remove directory. Use to delete directories. You must provide the name of the directory to be deleted. Can be given multiple directories at once. **Note:** only empty directories can be deleted.

## Commands for dealing with files

**cp**

Stands for copy. Takes two arguments: the file to be copied and the name or location of the new copy. If the second argument is the name of a directory, a file with the same name as the original is placed in the directory. Otherwise, the second argument is the name of the new file. Can be used to copy multiple files if the last argument is a directory. Note that you do not need to be in any particular directory to do a copy; you must simply specify enough information about the file to locate it.

**mv**

Stands for move. Can be used to rename a file or to move it from one directory to another. Like cp, takes two arguments: the name of the file to be moved and the new name or location of the file. Can be used to move multiple files if the last argument is a directory

## **rm**

Stands for remove. Can be used to delete files. On our system, it is set up to automatically confirm file deletion. That may not be the case on all systems you come in contact with. Confirm file deletion by typing **y** and enter in response to each query.

## **cat**

Lists the contents of the file argument to standard output (typically the screen, just as in your experience on Windows). If multiple filenames are provided, each file is listed in the order in which the files are given to the command.

## **more** or **less**

These commands can be used to browse text files a page at a time. **less** is a little more sophisticated than **more**. Both allow searching by typing **/** followed by the search pattern searching for the same pattern again by typing **n** moving a line at a time forward by pressing the enter key moving a page at a time forward by pressing the space bar moving a page at a time backward by typing **b** quitting by typing **q**

## **lpr**

This is the command you can use to print files. All files are printed on the printer in Old Union 131. You will need to enter your Sun logon on the print station next to the printer.

Example:

```
% lpr myclass.h myclass.cpp myprog.cpp
```

*This prints the three files named.*

## **Other useful details and commands**

\*

When specifying file names, it's often the case that you want to refer to all of the files in a directory or all of the files of a particular type. This can be done using **\***.

Examples:

```
% cp /home/mecalif/public/itk279/Program1/* .
```

This command will copy all files from the Program1 directory to the working directory. Note that the working directory has been specified using a single period, and that there is a space between the asterisk and the period.

```
% mv *.h *.cpp progs
```

This command will copy all files in the working directory that end in **.h** or **.cpp** into the subdirectory named **progs**.

## **man**

Stands for manual. When followed by a command, will show the manual page for the command. The man pages are somewhat challenging to read at first, but are useful for

finding out all the different options for a command or remembering exactly how to use a particular command.

Example:

% **man sort**

Will give information about the sort command and the various options that are possible for modifying the sorting.

### **grep**

This command is used for searching in files using a *regular expression*. For details on regular expressions, see the man page for **regex**. UNIX uses limited regular expressions for understanding command line arguments, which is why \* works as it does.

### **wc**

Stands for word count. Given a file name, it provides the number of lines, words, and characters in the file. Given multiple file names, it provides the data for each file, followed by a summary line with totals.

### **sort**

Can be used to sort files. Has a number of useful options. See the man page for details.

### **time**

Can be used to determine the time programs take to run. To use this command, you simply type “time” followed by the command to run the program. The time command give you four numbers: the user cpu time, the system cpu time, the wall time (i.e. the amount of actual time taken to run the program) and the percentage of the CPU that was allocated to the program over the time it ran. The number of primary interest to you will be the first number. The wall time is irrelevant, in general, because it varies with the load on the machine.

### **diff**

Allows you to compare two files to see if they’re the same. Useful for testing programs. Simply type diff filename1 filename2.

### **ctrl-c**

While you should always program very carefully, even the best of us occasionally manage to create a runaway program (infinite loops). If you need to make your program stop, the first thing to do is to type ctrl-c. That will almost always kill the program. If that does not work, open another window or create another connection to the **same** machine and use the **ps** command to find out what the *process id number* is for your runaway program. You may have to use the -a option to find your program. Once you know the process id, you can use the **kill** command to stop your program. You’ll use the -9 option for the kill command to guarantee that the process stops.

Example:

*Suppose that you have created a program named “fred” that is stuck in a loop and ctrl-c didn’t work. You open another window and try the ps command.*

% **ps**

```
PID TTY      TIME CMD
11327 pts/9    0:00 tcsh
```

*Since your program didn't show up, you try **ps -a**.*

```
% ps -a
```

```
PID TTY      TIME CMD
1048 pts/3    0:02 dtsessio
1343 pts/9    0:00 ps
11327 pts/9   0:00 tcsh
11468 pts/6    0:13 fred
11598 pts/3    0:00 more
```

*Here we see that the command **fred** has process id 11468, so we can now perform the kill command.*

```
% kill -9 11468
```

## Redirecting input and output

UNIX (like DOS) allows us to redirect input and output of commands. This can be useful in a variety of ways. It allows programs to be flexible, because they can work on standard input or write to standard output while allowing the user to store the data in files. We can even string commands together, sending the output of one command to the next command as input.

< redirects the input of a command to come from the following file. Often useful in running programs many times with the same input. I regularly use this technique for grading. ☺

Example:

```
% sicstus < runbufoidl
```

This will run sicstus (a Prolog system available on the Suns) and attach the file runbufoidl as standard input, so sicstus will execute the commands in the runbufoidl file. This input file is a standard text file, containing the information a user would type into the sicstus program.

> redirects the output a command to go to the following file. This can be useful for storing the output of a program in a file for later perusal.

Example:

```
% cat file1 file2 > file3
```

This will create a new file named file3 with the contents of file1 and file2.

| is called a pipe. It is used to pass the output of one command into the input of another command (through the pipe).

Example:

```
% grep abc testfile | wc
```

The grep command is executed, producing all of the lines in testfile that contain the characters abc; then those lines are passed as input to wc, which displays a count of the number of lines as well as the number of words and characters in those lines.

## Editing files in UNIX

There is a text editor available on the Sun desktop, but you will also need to use editors from home. There are three available to you.

### vi

This is an older editor. It has a lot of capabilities, but is somewhat clunky to use and rather cryptic. Not recommended unless you're trying to impress people with your esoteric knowledge.

### pico

This is the easiest of the three editors to use. Recommended for the majority of students. It displays the most likely commands at the bottom of the screen. Note that the ^ character is used to represent control throughout computer literature, so ^C is the same as control-c.

### emacs

This editor is less difficult than vi and by far the most powerful of the three editors. The serious UNIX user should definitely take the time and trouble required to learn emacs, particularly if he or she expects to work in a variety of programming languages (emacs has automatic formatting capabilities for many programming languages). Emacs does have a windowing interface. One of the best ways to start learning emacs is to sit down, start the editor by simply typing emacs at the command line, and then work through the tutorial. There are several resources on emacs around, including a good book published by O'Reilly (your professor has a copy of it she'd probably let you look at it if you ask nicely).

## Compiling C++ programs

To compile a C++ program, use the command `g++` followed by **all** of the `.cpp` files in the program. Note that the `.h` files are not included on the command line, only the `.cpp` files. When working on a program with multiple files, you may find it useful to create a makefile containing instructions on how to build the program. There are sample makefiles in my public area, and I will be glad to sit down with you and help you create your first makefiles and figure out how **make** works. You can also look at the **make** man page for information.

If you're writing programs of any size at all, it's a good idea to create a directory for each program. This makes compilation very easy, since the `*` works here, too.

### Example:

```
% g++ *.cpp
```

*This will correctly compile your program if all of the `.cpp` files in the working directory are part of your program.*

The executable file (the program you can actually run by typing the name of the program at the command prompt) produced by the command above is name **a.out**. This is the

default name for any C/C++ program you compile. However, you can specify a name for the program by using the `-o` option (the `o` stands for output).

Example:

```
% g++ -o myprog *.cpp
```

*Instead of creating an executable program named `a.out`, this creates an executable program named `myprog`.*

## **Using Blackboard to Submit**

To open a browser window, type `firefox&` at a command prompt. Once Firefox is running, use it as normal. You will find Blackboard at `blackboard.ilstu.edu`. The username and password are your ulid and associate password.

## Index of commands

<b>Name</b>	<b>Brief description</b>	<b>Page discussed</b>
<b>cat</b>	List contents of file	5
<b>cd</b>	Change directories	3
<b>cp</b>	Copy a file	4
<b>diff</b>	Compare two files	6
<b>dir</b>	List directory contents	4
<b>emacs</b>	A text editor	8
<b>g++</b>	C++ compiler	8
<b>grep</b>	Search files for expression	6
<b>kill</b>	Stop a UNIX process	6
<b>less</b>	Browse contents of a file	5
<b>logout</b>	Log out of UNIX	2
<b>lpr</b>	Print a file	5
<b>ls</b>	List directory contents	4
<b>man</b>	View manual page for a command	5
<b>mkdir</b>	Create a directory	4
<b>more</b>	Browse contents of a file	5
<b>mv</b>	Rename or move a file	4
<b>passwd</b>	Change UNIX password	2
<b>pico</b>	A text editor	8
<b>ps</b>	List current processes	7
<b>pwd</b>	Print current working directory	3
<b>rlogin</b>	Log on to another UNIX machine	1
<b>rm</b>	Delete a file	5
<b>rmdir</b>	Delete a directory	4
<b>sort</b>	Sort data	6
<b>time</b>	Times a program	6
<b>vi</b>	A text editor	8
<b>wc</b>	Count words in a file	6