

Harry McGregor — CIS137 — Introduction to the UNIX  
Operating System

Original version ©Louis Taber, PCC

May 28, 2001

# Contents

<b>1</b>	<b>CIS137 Class information</b>	<b>4</b>
<b>2</b>	<b>Syllabus</b>	<b>4</b>
2.1	Catalog Description . . . . .	4
2.2	Textbooks . . . . .	4
2.3	Class content . . . . .	5
2.4	Lab resources . . . . .	6
<b>3</b>	<b>Attendance and Grading Policy</b>	<b>6</b>
3.1	Course grade . . . . .	6
3.2	Attendance . . . . .	7
3.3	Quizzes . . . . .	7
3.4	Exams . . . . .	7
3.5	Labs . . . . .	7
3.6	Software Copyright and License . . . . .	7
3.7	Americans with Disabilities Act . . . . .	7
<b>4</b>	<b>A little bit of background</b>	<b>8</b>
4.1	What is Unix . . . . .	8
4.2	Early History of Unix . . . . .	8
4.3	MIT Project MAC . . . . .	8
4.4	Digital Equipment Corporation . . . . .	9

4.5	Unix Philosophy . . . . .	9
4.6	UC Berkeley - Unix development in higher education . . . . .	11
4.7	Comercialization - Unix in Corporate America . . . . .	11
4.8	The Free Software Foundation and the GNU project . . . . .	12
4.9	The Linux kernel, and a GNU Operating System . . . . .	12
<b>5</b>	<b>Unix "man" pages</b>	<b>15</b>
5.1	What are "Man Pages" . . . . .	15
5.2	How to access man pages . . . . .	15
5.3	Man page sections . . . . .	16
5.3.1	Section 1 . . . . .	17
5.3.2	Section 2 . . . . .	17
5.3.3	Section 3 . . . . .	17
5.3.4	Section 4 . . . . .	18
5.3.5	Section 5 . . . . .	18
5.3.6	Section 6 . . . . .	18
5.3.7	Section 7 . . . . .	18
5.3.8	Section 8 . . . . .	18
5.4	Layout of man pages . . . . .	18
5.5	Section Introduction . . . . .	19
<b>6</b>	<b>Labs</b>	<b>20</b>
6.1	Setup UNIX account . . . . .	20

6.2	Some UNIX commands . . . . .	22
6.3	Editor 1 – About Me . . . . .	27
6.4	First Shell Script . . . . .	29
6.5	Editor – vi . . . . .	33
6.6	Editor 3 – emacs . . . . .	38
6.7	grep and regular expressions . . . . .	40
	6.7.1 sh and csh rules . . . . .	41
	6.7.2 regular expression rules . . . . .	42
	6.7.3 extended regular expression rules . . . . .	43
6.8	Shell script — dir . . . . .	43
6.9	Three <b>bash</b> scripts . . . . .	45
6.10	“C” Programming Language . . . . .	46
6.11	Menu shell script . . . . .	49
6.12	Internet . . . . .	51
6.13	Text processing - HTML . . . . .	52
6.14	bc - Extended Precision Calculator . . . . .	53
6.15	sed - A Stream Editor . . . . .	57
6.16	awk . . . . .	60
6.17	Perl . . . . .	68
6.18	System Administration . . . . .	69

[31] Sams publishing  
UNIX Unleashed  
?????? with CD about \$50.00

[32] Shaw, Myril Clement & Shaw, Susan Soltis  
*UNIX Internals: A System Operations Handbook*  
Tab Books Inc. 1987 ISBN 0-8306-2951-3 (paperback)

[33] Stevens, W. Richard  
*Advanced Programming in the UNIX Environment*  
Addison-Wesley Publishing Company 1992 ISBN 0-201-56317-7

[34] Todino, Grace; Starang, John; & Peak, Jerry  
*Learning the Unix Operating System, Third Edition*  
O’Reilly & Associates, Inc. (info@ora.com) \$9.95 Copyright 1993. ISBN 1-56592-060-0

[35] Tophan, Douglas W.  
*Portable UNIX: Key Commands at Your Fingertips!* Wiley Copyright 1992  
ISBN 0-471-57926-2 \$18.95  
This is a summary book. Commands are sorted by function. At 274 pages the descriptions are quite short. The only text editor the book covers is vi. Chapters for the Bourne, “C”, and Korn shells.

[36] Larry Wall & Randy L. Schwartz  
*Programming perl*  
O’Reilly & Associates, Inc. (info@ora.com) Copyright 1992 ISBN 0-937175-64-1

[37] Welsh, Matt & Kaufman, Lar  
*Running Linux, A UNIX-compatible operating system for the PC*  
O’Reilly & Associates, Inc. (info@ora.com) 1995 ISBN 1-56592-100-3 \$24.95

[38] Gancarz, Mike  
*The UNIX Philosophy*  
Butterworth-Heinemann Copyright 1994 ISBN 1555581234 \$26.95

- [21] Kernighan, Brian W. & Pike, Rob  
*The UNIX programming environment*  
 Prentice-Hall, c1984 ISBN 0-13-937681-X (pbk.)
- [22] Kernighan, Brian W. & Ritchie, Dennis M.  
*The C Programming Language, Second edition*  
 Prentice-Hall, c1988 ISBN 0-13-110362-8 (pbk.)
- [23] Krol, Ed  
*The Whole Internet*  
 O'Reilly & Associates, Inc. (info@ora.com) 1992 ISBN 0-937175-84-6 \$27.95
- [24] Leffler, Samuel J.; McKusik, Marshall Kirk; Karels, Michael J; & Quarterman, John S.  
*The Design and implementation of the 4.3BSD UNIX operating system*  
 Publisher: Addison-Wesley, c1989. 471 p. ISBN: 0-201-06196-1
- [25] Leffler, Samuel J. & McKusik, Marshall Kirk  
*The Design and implementation of the 4.3BSD UNIX operating system, Answer Book*  
 Publisher: Addison-Wesley, c1991. ISBN: 0-201-54629-9
- [26] Nemeth, Evi; Snyder, Garth; & Seebass, Scott  
*Unix System Administration Handbook*  
 Prentice-Hall 1989 ISBN 0-13-933441-6
- [27] Matthew, Neil & Stones, Richard  
*Beginning Linux Programming*  
 WROX press Copyright 1996. ISBN 1-874416-68-0
- [28] Plauger, P.J.  
*The Standard C Library*  
 Prentice-Hall 1992. ISBN 0-13-133509-9
- [29] Ritchie, D.M. & Thompson, K.  
*The UNIX Timesharing System* The Bell System Technical Journal  
 Vol. 57, No. 6, July-August 1978, p1905-1929  
 AT&T Description of UNIX
- [30] Salus, Peter H.  
*A Quarter Century of Unix*  
 Addison-Wesley 1994 ISBN 0-201-54777-5 \$24.75 QA76.76 .O63S342 1994

## 1 CIS137 Class information

This document is the syllabus, labs, attendance and grading policy, and most of the handouts used in Harry McGregor's CIS137 class.

The hypertext version of this document contains many links to other references.

I would like to thank John Skapura and Louis Taber of the Computer Information Systems Department at Pima Community College, West Campus, and many unnamed students for their help and assistance in getting the materials ready for my class and helping my class run smoothly.

This document is written in a version of  $\LaTeX$  called HyperLatex written by Otfried Cheong. Hyper $\LaTeX$  uses Leslie Lamport's  $\LaTeX$ , a document preparation system, Donald E. Knuth's  $\TeX$ , a typesetting program, and a text editor.

This document is available under a FSF (<http://www.fsf.org>) license. Please contact the original author (Louis Taber, [ltaber@lt.tucson.az.us](mailto:ltaber@lt.tucson.az.us)) if you are interested.

The **most current** version of this document can be found at URL <http://www.osef.org/harry/cis> . Links to sections.

## 2 Syllabus

### 2.1 Catalog Description

Prerequisites: CIS135 or consent of the instructor.

Principles and tools of the UNIX operating system. Includes utilities, file structure, text editors, tools, documentation, networking, and the comparison and usage of different shells.

### 2.2 Textbooks

- Learning the Unix Operating System, Third Edition [34]
- Unix in a Nutshell - For System 5 and Solaris 2.0 [15]

## 2.3 Class content

- History and importance of UNIX
- Logging on/off  
.profile, .cshrc, .login, .logoff
- Passwords, security, file protection  
passwd
- Regular expressions and extended regular expressions  
grep, egrep, ex, vi, emacs, awk, perl
- Utilities  
cat, less, more, head, tail, ls, cd, mkdir, rmdir, pwd, who, w, cp, ln, mv, rm, compress, gzip, gunzip, cut, lpr, lpq, lpc, lprm, date, cal, wc, echo, du, df, test, expr, man, info, chmod, chown, bc, dc, split, paste, file
- Editors  
ex, ed, vi, emacs, xedit, pico, .emacs, .exrc, sed
- File structure  
mount, umount, /dev, /etc/exports, /etc/fstab
- Shell & Shell scripts  
bourne (sh), “C” (csh), korn, bash, tcsh  
argument substitution, wildcards, meta characters, file name completion, and history
- Text formatting  
nroff & troff (manual pages), HTML (www)
- “C” programming and “C” language interface to UNIX OS
- Perl & CGI
- System administration, Internal structure  
/etc, /etc/passwd, ps, netstat, termcap, printcap, rc, rc.local, crack, shutdown, sync, run levels, disk partitions, sudo, su
- Communication  
Mail, pine, MH, talk, wall, write
- Internet TCP/IP  
ftp, telnet, finger, ping, nslookup, traceroute, httpd, ~/public.html

- [11] Evens, Andrew; Matthew, Neil; & Stones, Richard  
*Instant UNIX*  
WROX press Copyright 1995. ISBN 1-874416-65-6
- [12] Gancarz, Mike  
*The UNIX Philosophy*  
Digital Press 1995 ISBN 1-55558-123-4 QA76 .76 .O63 G365 1995
- [13] Gianone, Christine M.  
*Using MS-DOS Kermit: Connecting your PC to the Electronic World, 2ed*  
Copyright 1992. ISBN 1-55558-082-3
- [14] Glass, Graham  
*UNIX for Programmers and Users: A complete Guide*  
Prentice-Hall ISBN 0-13-480880-0
- [15] Daniel Gilly and the staff of O’Reilly & Associates  
*Unix in a Nutshell - For System 5 and Solaris 2.0*  
O’Reilly & Associates, Inc. (info@ora.com) \$9.95  
Copyright 1993. ISBN 1-56592-001-5
- [16] Gundavaram, Shishir  
*CGI Programming*  
O’Reilly & Associates, Inc. (info@ora.com)  
Copyright 1996, ISBN 1-56592-168-2 \$29.95
- [17] Hall, Mark & Barry, John  
*Sunburst: The Ascent of Sun Microsystems*  
Contemporary Books, Chicago Copyright 1990 ISBN 0-8092-3989-2
- [18] Harbison & Steele  
*C: A reference manual*  
Prentice-Hall ISBN 0-13-326224-3
- [19] Hardin, Garrett James, 1915-  
*Living within limits: Ecology, economics, and population taboos*  
Oxford University Press, 1993.  
ISBN 019507811X (acid-free paper) \$25.00
- [20] Hunt, Craig  
*TCP/IP Network Administration*  
O’Reilly & Associates, Inc. (info@ora.com) 1992 ISBN 0-937175-82-X

- [2] AT&T 1-800-432-6600 \$35.06 307-075  
*UNIX System V/386 Release 3.2 Programmers Reference Manual*  
 “Not all commands are available on every UNIX system and some features may require additional utilities“ System calls, functions and software tools only.
- [3] Bach, Maurice J.  
*Design of the UNIX Operating System*  
 Prentice-Hall Copyright 1986 ISBN 0-13-201799-7
- [4] Cameron, Debra & Rosenblatt, Bill  
*Learning GNU Emacs*  
 O’Reilly & Associates, Inc. (info@ora.com) \$27.95  
 Copyright: 1991 ISBN 0-937175-84-6
- [5] Comer, Douglas E  
*Internetworking with TCP/IP, Volume 1*  
*Principles, Protocols, and Architecture*  
 Prentice Hall 1991 ISBN 0-13-468505-9
- [6] Comer, Douglas E & Stevens, David L.  
*Internetworking with TCP/IP, Volume 2*  
*Design, Implementation, and Internals*  
 Prentice Hall 1991
- [7] Comer, Douglas E & Stevens, David L.  
*Internetworking with TCP/IP, Volume 3 BSD Socket Version*  
*Client - Server Programming and Applications*  
 Prentice Hall 1993 ISBN 0-13-474222-2
- [8] Comer, Douglas E & Stevens, David L.  
*Internetworking with TCP/IP, Volume 3 AT&T TLI Version*  
*Client - Server Programming and Applications*  
 Prentice Hall 1993 ISBN 0-13-474230-3
- [9] da Cruz, Frank & Gianone, Christine M.  
*Using C-Kermit*  
 Digital Press 1-800-332-5656 x937
- [10] Doughery, Dale  
*sed & awk*  
 O’Reilly & Associates, Inc. (info@ora.com) \$27.95 Copyright 1990. ISBN 0-937175-59-5

This list is a sampling of the many possibilities. The exact content of the class will depend on the interests of the students enrolled and the time available.

## 2.4 Lab resources

We will be using gort running Linux (Debian 2.2 - potato) Kernel 2.4.2 on an IBM/PC compatible Pentium II PC with 384 Megabytes of RAM and 27 Gigabytes of disk storage. The system is internet connected. (gort.cscwc.pima.edu, 144.90.66.69) If you have access to a UNIX based system you will be able to do most of the labs on it. The system has OpenSSH installed for secure shell connections.

## 3 Attendance and Grading Policy

### 3.1 Course grade

Your course grade will be based on the average of the best two scores out of three different class measurements. These measurements are:

1. quiz performance
2. lab performance
3. final exam performance

I may, at my discretion, add a fixed, positive percentage to all student’s scores in a class.

Your course grade will be determined by using the average of your two highest performance measurements and a possible class adjustment.

Greater than 90%	A
Greater than 80% and less than 90%	B
Greater than 70% and less than 80%	C
Greater than 45% and less than 70%	D
Greater than 40% and less than 45%	F
Less than 40%	Y

### 3.2 Attendance

Your attendance in class will help you learn the material covered. Some times I will take attendance. Non-attendance and late arrival may result in missed quizzes. If you expect to miss two or more consecutive classes, please contact me by phone or email.

### 3.3 Quizzes

Quizzes help both the instructor and the students know if the material needs to be reviewed. There will be unannounced quizzes. Quizzes will vary in weight.

### 3.4 Exams

The only exam in the class will be the final exam.

### 3.5 Labs

Labs should be completed in class. If you prefer to complete the labs at home, it is recommended that the labs be turned in on the due date (end of each week). No labs will be accepted for evaluation after the final due date at the end of the semester.

### 3.6 Software Copyright and License

Some of the software used in my classes is copyrighted and is licensed to Pima Community College. Copying of licensed and/or copyrighted software is not permitted. (Copying of some software can be desirable and encouraged. Check the license agreement.)

### 3.7 Americans with Disabilities Act

Pima County Community College District strives to comply with the provisions of Title III of the Americans with Disabilities Act of 1990 and Section 504 of the Rehabilitation Act of 1973. Students with disabilities requiring special accommodations

```
find /usr/bin -user root -perm -4000 -print
```

Append this file to your printout.

- System maintenance

1. Use the following command to copy a set of files keeping the existing permission bits.

```
cd
mkdir from-dir
mkdir to-dir
cd from-dir
** Put some files in the from-dir directory **
tar -cf - . | ( cd to-dir; tar xvf - )
```

**Make sure that the *to-dir* is not in the *from-dir*.** You may want to remove both *from-dir* and *to-dir*.

2. Use the `dmesg` command to see the “power-up” messages for the system. Include this information in your “turn-in” file.
3. Clean up your section of the file system by removing unwanted files and directories. You can use `rm -r backup-dir` to remove all of the files and the new `backup-dir` sub-directory. Keep your floppy as long as you want it. Also delete `/var/tmp/your-user-name`.

Turn in the printout and mark it with:

```
your-name
McGregor CIS137
Lab 6.18: System Administration
```

and the results of the file compression.

## References

- [1] Aho, Alfred V., Kernighan, Brian W. & Weinberger, Peter J.  
*The AWK programming language*  
Addison-Wesley Pub. Co., Reading, Mass.  
Copyright: 1988 ISBN 020107981X (pbk.) \$21.95

```
your-user-name.tar
and
your-user-name.tar.gz
```

Write the result of your comparison on your final output.

- (Optional) Copy this to a floppy disk using `Kermit` or `ftp`. There are PC systems in the computer lab (TBA) that can be used to do this, or you can do it over the Internet. Make sure that the file is transferred as a binary file.
- Create a new subdirectory in your area (called “`backup-dir`”). Uncompress your file into this area, then expand the `tar` file to restore your files. The first `cd` will take you back to your home directory.

```
cd
mkdir backup-dir
cd backup-dir
tar -xvzf /var/tmp/your-user-name.tar.gz
```

The `-x` option is for “eXtract”. The `-v` option is for verbose. It will list the file names. The `-z` option will decompress the tar file on the fly. The `-f` option is for a following file name. `tar` defaults to a tape drive.

- Save a copy of a recursive directory to be turned in. For this lab turn in all of the requested commands together as a single printout.

```
cd
ls -lR
```

This should include your `backup-dir` and the original subdirectory. Remember to print all of your results together as a single printout for this lab.

- Monitoring the System

- Use the `df` command to look at the file systems on gort. Save a copy of the results.
- Use the `mount` command to look at the file systems mounted on the system. Append this to your output.
- Use `finger`, `ps -ae`, and `ps -al` to see what else is on the system. `ps` is a process status program. It shows what you and other people have running. Append the results of these commands to the file from the previous step.
- Run the `top` command. This shows the top processes on the system. End the command with a `q`.
- Look for `suid` programs owned by root.

are strongly encouraged to notify the instructor at the **beginning** of the semester so that appropriate verification and identification of reasonable accommodations may be made in a timely manner. (Accommodations cannot be made without verification of need.)

## 4 A little bit of background

### 4.1 What is Unix

Unix is a multiuser operating system capable of dealing with hundreds or thousands of simultaneous users. Unix is a very robust and scaleable operating system, capable of running everything from prototype wrist watch computers to some of the worlds largest and fastest super-computers.

### 4.2 Early History of Unix

In 1969 two researchers at AT&T Bell Labs developed the Unix computer operating system. The development of Unix has been largely regarded as the single most important software invention to come out of Bell Labs. These two researchers were Ken Thompson and Dennis Ritchie.

### 4.3 MIT Project MAC

AT&T, GE, IBM, and MIT’s project MAC (Multiple Access Computers) joined to develop the time-sharing system, MULTICS (Multiplexed Information and Computing Services). Development started in 1965.

In 1966 Ken Thompson finished his studies at University of California, Berkeley (UC Berkeley), and joined the technical staff at Bell Labs. Mr. Thompson was set to work on MULTICS.

In 1968 Dennis Ritchie stopped work on his Ph.D. at Harvard, and joined Bell Labs to work on the MULTICS project. Unfortunately, only a few months after this, Bell Labs dropped out of the MULTICS project due to the project not fulfilling it’s objectives. The MULTICS project was envisioned to be able to support up to 1,000 online users. At that time MULTICS could barely support three users.

Just after AT&T left the MULTICS project, Ken Thompson developed a new idea for a type of file system, and started discussing it with Dennis Ritchie and Rudd Canaday. Thompson wrote the first version of UNICS for the Digital Equipment Corporation (DEC) PDP-7 in one month, while his wife was on vacation. He allocated one week per part of the operating system, one for the kernel, one for the shell, one for the editor, and one for the assembler.

The name UNICS was a pun on MULTICS, and stood for Uniplexed Information and Computing Services. Soon after this, the spelling was changed to Unix, which is not an acronym, just a name.

#### 4.4 Digital Equipment Corporation

In 1970 Digital Equipment Corporation (DEC) began shipping the PDP-11. This was the first mass produced 16bit computer, and it revolutionized the industry. Many Universities purchased systems for computer science research. Also DEC gave a number of Universities (MIT for example) PDP class systems for research purposes. Bell Labs purchased a PDP-11 for text processing in the legal department. The system was the first production system at Bell Labs to be implemented with the Unix operating system.

#### 4.5 Unix Philosophy

By 1973 pipes were invented, and the Unix philosophy begins to emerge. The key aspect of the Unix philosophy was to write programs that do one thing, and do it well. Pipes, the ability to link the output of one program to the input of another program helped make the small program philosophy successful. Pipes make it possible for example, to use one program that archives files (tar) and another program that compresses files (compress or gzip) linked together to create a compressed archive.

The following principles have been extracted from the book The Unix Philosophy [12], which I would recommend to anyone who wants to understand why Unix is useful and powerful.

- Small is beautiful
- Small functional programs are readily understood, applied, and reused, and there is not as much complexity amongst which bugs can hide.

are interested in Unix/Linux Systems Administration, please look in the PCC class, CIS 225, Linux Networking. I will be teaching this class during the fall semester at the West Campus.

If you are using gort for the system administration lab I would like you to produce a single annotated printout. So as you go through the following steps save your output into a file that you can edit later. One way of doing this is by using the `script` command. It will produce a file called *typescript*.

- Backup and Restore

1. Look in your area with the `du` command to find a subdirectory with less than 1 Mbyte of data.
2. Use `tar` to create a file that contains all of the files in your <1Mbyte area. The command should look something like this:

```
cd /var/tmp/  
tar -cvf your-user-name.tar ~/
```

This will create a `tar` file in `/var/tmp/` with your user-name as the file name. `tar` will try to `tar` its own output if given the chance.

3. Do a

```
ls -l
```

to find out the size of the archive file **prior** to the compression. Write down the size of the file; it will be deleted by the compress command. Compress the files using the `compress` command:

```
compress your-user-name.tar
```

Compare the size of  
`your-user-name.tar`  
and  
`your-user-name.tar.Z`.

Write this on your final printout. Write down the compression percentage.

4. Uncompress the file.  

```
uncompress your-user-name.tar.Z
```
5. Some systems might not have the `gzip` and `gunzip` commands. Skip this section if you must. Re-compress the files using the `gnu gzip` command:  

```
gzip your-user-name.tar
```

Compare the size of

- `s2p` — for converting `sed` programs to `perl`
- `a2p` — for converting `awk` programs to `perl`

In this lab I would like you to:

1. Create the directory `cgi-bin` in your existing directory `~/public.html/`.
2. Make this new directory your working directory.
3. copy over `perl.cgi` from the from `/home/cis137/perl.cgi`

```
cp /home/cis137/perl.cgi .
```

4. Place `perl.cgi` and the `sed` and `awk` data files in your directory `public.html/cgi-bin` directory.
5. Set the permissions so that you can run the program as a CGI script. Set the data files to world readable. When the program is run it will not be run as you.
6. Modify the program so that it does the last part of the `awk` lab, NOT the `awk` program that I wrote.
7. Add your name and the class number, “CIS137”, to your output.
8. Turn in a printed output of your results from a web browser. The URL to access your page will be:

```
http://gort.cscwc.pima.edu/~user-name/cgi-bin/perl.cgi
```

```
your-name
McGregor CIS137
Lab 6.17: perl
```

## 6.18 System Administration

The term “system administration” covers a lot of areas. If you are a good Unix systems administrator, you have many different tasks to deal with and you have the root password. On gort, students don’t know the root password, so what can be done is limited. This lab is only a brief introduction to System Administration. If you

- Make each program do one thing and do it well
- Multiple useful programs can then be composed together to perform more complex functions.
- Prototype as soon as possible
- Early affirmation/denial of the validity of assumptions is really valuable.
- Choose portability over efficiency
- Effectiveness beats efficiency. If software works adequately, it will work even better on next year’s hardware.
- Store numerical data in flat ASCII files
- Portable data is as important as portable code... Using text data files makes it easy to use powerful text-processing tools to manipulate the data.
- Don’t force yourself or others to reimplement functionality; make it easy to reuse functionality.
- Use shell scripts to increase leverage and portability
- Shell scripts don’t need to be compiled or recompiled
- Avoid captive user interfaces
- Make every program a filter
- Allow the user to tailor the environment
- Make OS kernels small and lightweight
- Use lower case and keep it short
- Keep text online, and use powerful tools to manipulate it.
- Silence is golden
- Think parallel
- SMP and networked parallel processing are easier to harness if we have a bunch of small pipe-connected components, rather than a single large monolithic process.
- The sum of the parts is greater than the whole

- A set of programs that can be composed in unexpected ways is far more flexible than one monolithic one.
- Look for the 90% solution - Solving the next 5% probably costs more than the previous 90%; solving the next 5% after that costs more still.
- The notion of directory and file hierarchies is really powerful, and can be extended to the naming of devices, network services, graphical resources...

#### 4.6 UC Berkeley - Unix development in higher education

UC Berkeley has been a major development center for Unix. UC Berkeley's involvement started in 1973 when Professor Bob Fabry attended the Symposium on Operating Systems Principles at Purdue University. Ken Thompson and Dennis Ritchie presented their first paper on the Unix operating system at this symposium. In 1974 Professor Fabry was able to attain the necessary funds to purchase a PDP-11/45, and with the remote assistance of Ken Thompson, Unix was installed on the system. To gain the funds, the Computer Science, Mathematics, and Statistics departments combined their available funds. In late 1975 Ken Thompson took a one year sabbatical from Bell Laboratories and returned to UC Berkeley as a visiting Professor. Work on Unix at UC Berkeley centered around the Defense Advanced Research Projects Agency (DARPA), and the need for an operating system for the ARPA network, which has today grown into the InterNet.

#### 4.7 Comercialization - Unix in Corporate America

During the late 1970s and early 1980s several companies formed around the Unix operating system. Many of these companies are still leaders in the computer industry today, these include Sun Microsystems(<http://www.sun.com>), Silicon Graphics Inc., and The Santa Cruz Operation, Inc. Many other companies in the industry distributed Unix, in fact Microsoft had their own distribution of Unix, Zenix. Unix based companies are the companies setting the standard for server performance and reliability. For example Sun Microsystems servers, combined with SUN's Solaris Unix operating system, are the preferred platform for Internet Service Providers.

the *AWK* book.

6. Write an **awk** program that prints the average of the third field in *data* file.

7. Write an **awk** program that removes the first field and prints only those lines where the third field is greater than 15 million. (Again, use the *data*).

**Turn in this page.**

#### 6.17 Perl

**perl** is a programming language developed and supported by Larry Wall currently employed by O'Reilly & Associates, Inc.. It is an interpreted programming language with good string handling capabilities. It accounts for a lot of the CGI programming that is done on the WWW.

Look at these two books:

- Programming in perl [36] and
- CGI Programming [16].

You may also want to look at the URLs:

- <http://www.ora.com/publishing/perl/resource.htm>.
- <http://www.cpan.org>.

There are two programs you may want to look at:

Table 11: Flow of control

```

if(expression) statement
if(expression) statement1 else statement2
while( expression) statement
for(expression1;expression2;expression3)statement
do statement while( expression )
break
continue
next
exit
exit expression
return

```

Table 12: AWK escape sequences

Escape Sequence	Meaning
<code>\b</code>	backspace
<code>\f</code>	form feed
<code>\n</code>	new line — ASCII lf
<code>\r</code>	carriage return — ASCII cr
<code>\t</code>	Horizontal tab — ASCII tab
<code>\nnn</code>	ASCII octal value
<code>\\</code>	ASCII backslash
<code>\c</code>	For any character <i>c</i>

## 4.8 The Free Software Foundation and the GNU project

The GNU project, started by Richard M. Stallman, is based on the idea of freedom of information and sharing of knowledge. While many people in the computer industry today have been keen on protecting intellectual property, for the developer, the GNU project has made a name for itself, by releasing licenses that protect the intellectual property itself, and protect that protect the users of the intellectual property. GNU stands for GNU's Not Unix.

The Free Software Foundation (fsf), the host company of the GNU project, was founded on the idea that software, like knowledge, should be freely available for anyone, not just to use, but to modify.

Over the years o

f the GNU project, most parts of the Unix operating system had been re-implemented, in an open, GPLed fashion. Unfortunately the core component of a GNU operating system was missing, the kernel. The kernel of an operating system interfaces with the computer hardware and is the underpinning of the system.

## 4.9 The Linux kernel, and a GNU Operating System

In 1991 Linus Torvalds was a computer science student at the University of Helsinki in Finland. He was in search of a Unix system he could run on his home computer, to complete assignments for class. At the time the only version of Unix available for the Intel 386 cpu, was a Minix. While Minix was alright, Linus wanted something differnt, and he decided to write his own kernel, to combine with the GNU project's utilities. In October of 1991 Linus released version 0.02 onto the internet, via the minix usenet group (comp.os.minix). The combination of the Linux kernel, written by Linus, and improved upon by thousands of programers around the world, and the GNU project, is the GNU/Linux operating system.

Here is a copy of two messages posted by Linus:

```

First message, pre Linux 0.01
From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Summary: small poll for my new operating system
Message-ID: j1991Aug25.205708.9541@klaava.Helsinki.FIj

```

Date: 25 Aug 91 20:57:08 GMT  
Organization: University of Helsinki

Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

Linus (torvalds@kruuna.helsinki.fi)

PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT portable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-).

Second message, Linux 0.02

From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)  
Newsgroups: comp.os.minix  
Subject: Free minix-like kernel sources for 386-AT  
Message-ID: j1991Oct5.054106.4647@klaava.Helsinki.FI;  
Date: 5 Oct 91 05:41:06 GMT  
Organization: University of Helsinki

Do you pine for the nice days of minix-1.1, when men were men and wrote their own device drivers? Are you without a nice project and just dying to cut your teeth on a OS you can try to modify for your needs? Are you finding it frustrating when everything works on minix? No more all-

Table 10: Operators — Logical and Conditional

Relational Operators	
<	Less than
<=	Less than or equal
==	Equal
!=	Not equal
>=	Greater than or equal
>	Greater than
~	Does the string contain the re
!~	Does the string not contain the re

  

Logical Operators	
	Logical OR
&&	Logical AND
!	Logical NOT

3. What `awk` command will print out all lines in `args.c` that are longer than 20 characters?

4. What `awk` command will print out lines in `args.c` that don't have a "(" and are longer than 10 characters?

5. Using the input file `~cis137/data`

```
Arizona Phoenix 6285295 Tucson 32
California Sacramento 38593635 Eureka 12
Oregon Salem 12345234 Portland 45
Washington Olympia 6549872 Seattle 36
Illinois Springfield 6759346 Chicago 14
Maine Augusta 456923 Lewiston 23
Texas Austin 23967433 Houston 26
```

Write an `awk` program that reverses the order of the fields. Note that this file always has 5 fields. This does not need to be as complicated as the example in

Table 9: Operators — Arithmetic

Arithmetic Operators	
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder
^	Exponentiation
Assignment Operators	
=	Assignment
+=	Addition and assignment
-=	Subtraction and assignment
*=	Multiplication and assignment
/=	Division and assignment
%=	Remainder and assignment
^=	Exponentiation and assignment
Increment & Decrement Operators	
++	increment (prefix & postfix)
--	decrement (prefix & postfix)

Lab 6.16: `awk` McGregor CIS137  
**your name:**

1. What `awk` command can you use to print out all lines in `args.c` that have `printf`?

2. What `awk` command will print out all lines in `args.c` that don't have a plus sign?

nigheters to get a nifty program working? Then this post might be just for you :-)

As I mentioned a month(?) ago, I'm working on a free version of a minix-lookalike for AT-386 computers. It has finally reached the stage where it's even usable (though may not be depending on what you want), and I am willing to put out the sources for wider distribution. It is just version 0.02 (+1 (very small) patch already), but I've successfully run `bash/gcc/gnu-make/gnu-sed/compress` etc under it.

Sources for this pet project of mine can be found at `nic.funet.fi` (128.214.6.100) in the directory `/pub/OS/Linux`. The directory also contains some README-file and a couple of binaries to work under linux (`bash`, `update` and `gcc`, what more can you ask for :-). Full kernel source is provided, as no minix code has been used. Library sources are only partially free, so that cannot be distributed currently. The system is able to compile "as-is" and has been known to work. Heh. Sources to the binaries (`bash` and `gcc`) can be found at the same place in `/pub/gnu`.

ALERT! WARNING! NOTE! These sources still need minix-386 to be compiled (and `gcc-1.40`, possibly `1.37.1`, haven't tested), and you need minix to set it up if you want to run it, so it is not yet a standalone system for those of you without minix. I'm working on it. You also need to be something of a hacker to set it up (?), so for those hoping for an alternative to minix-386, please ignore me. It is currently meant for hackers interested in operating systems and 386's with access to minix.

The system needs an AT-compatible harddisk (IDE is fine) and EGA/VGA. If you are still interested, please ftp the README/RELNOTES, and/or mail me for additional info.

I can (well, almost) hear you asking yourselves "why?". Hurd will be out in a year (or two, or next month, who knows), and I've already got minix. This is a program for hackers by a hacker. I've enjoyed doing it, and somebody might enjoy looking at it and even modifying it for their own needs. It is still small enough to understand, use and modify, and I'm looking forward to any comments you might have.

I'm also interested in hearing from anybody who has written any of the utilities/library functions for minix. If your efforts are freely distributable (under copyright or even public domain), I'd like to hear

from you, so I can add them to the system. I'm using Earl Chews estdio right now (thanks for a nice and working system Earl), and similar works will be very wellcome. Your (C)'s will of course be left intact. Drop me a line if you are willing to let me use your code.

Linus

PS. to PHIL NELSON! I'm unable to get through to you, and keep getting "forward error - strawberry unknown domain" or something.

During the past 9 years the popularity of Linux grew from a small group of computer enthusiasts to include many major corporations. Linux and the GNU project are bringing back the idea of shared computer source code, and cooperation between competitors.

Many computer companies are realizing that they can reduce costs by sharing the same (free) operating system, instead of devoting programers time to development of their own proprietary operating systems.

## 5 Unix "man" pages

### 5.1 What are "Man Pages"

On-line manual pages, called "man pages", have for a long time been the only available documentation on Unix systems. Man pages have a long-standing reputation for being hard to read, confusing, and in general inadequate for beginners, because they are designed as a reference rather than an instruction manual. However, even today they represent the most complete documentation available.

### 5.2 How to access man pages

The command for accessing man pages is "man". In general you will type "man topic", but only if you already know the exact topic name. (Almost) all unix commands are topics, so for instance "man ls" provides information on using the ls command.

Table 8: String Functions

<code>gsub(r,s)</code>	Globally substitute <i>s</i> for <i>r</i> in <i>\$0</i> returns number of substitutions
<code>gsub(r,s,t)</code>	Globally substitute <i>s</i> for <i>r</i> in <i>t</i> returns number of substitutions
<code>index(s,t)</code>	Return first position of <i>t</i> in <i>s</i> returns number of substitutions
<code>length(s)</code>	Returns length of <i>s</i>
<code>match(s,re)</code>	Test <i>s</i> for regular expression <i>re</i> Returns index or 0
	Sets RSTART and RLENGTH
<code>split(s,a)</code>	split <i>s</i> into array <i>a</i> on FS Returns number of fields returned
<code>split(s,a,fs)</code>	split <i>s</i> into array <i>a</i> on <i>fs</i> Returns number of fields returned
<code>sprintf(format,list)</code>	return <i>list</i> formatted by <i>format</i>
<code>sub(r,s)</code>	Substitute <i>s</i> for longest left-most substring in <i>\$0</i>
<code>sub(r,s,t)</code>	Substitute <i>s</i> for longest left-most substring in <i>t</i>
<code>substr(s,p)</code>	Return substring of <i>s</i> starting at <i>p</i> to end
<code>substr(s,p,n)</code>	Return substring of <i>s</i> starting at <i>p</i> of length <i>n</i>

Table 6: Built-in variables

Variable	Meaning	Default
ARGC	Number of command line arguments	—
ARGV	Array of command line arguments	—
FILENAME	Name of current input file	—
FNR	Record number of current file	—
FS	Input field separator	" "
NF	Number of fields in current record	—
NR	Number of records read	—
OFMT	Output format for numbers	"%.6g"
OFS	Output field separator	" "
ORS	Output record separator	"\n"
RLENGTH	Length of string matched with function <code>match</code>	—
RS	Input record separator	"\n"
RSTART	Start of string matched with function <code>match</code>	—
SUBSEP	Subscript separator	"\034"

Table 7: Arithmetic Functions

<code>atan2(y, x)</code>	arctangent of $y/x$ in the range of $-pi$ to $pi$
<code>cos(x)</code>	cosine of $x$ ( $x$ in radians)
<code>exp(x)</code>	$e^x$
<code>int(x)</code>	integer part of $x$
<code>log(x)</code>	logarithm base $e$ of $x$
<code>rand()</code>	random number from 0 to 1 ( $0 \leq \text{rand}() < 1$ )
<code>sin(x)</code>	sine of $x$ ( $x$ in radians)
<code>sqrt(x)</code>	square root of $x$
<code>srand(x)</code>	$x$ is seed for <code>rand()</code>

Of course, if you want to find the command to perform some specific task, you can see that this isn't too useful. You can hunt for subjects using "man -k subject". For instance, "man -k permission" will show you a listing of just a few man pages, including one for `chmod` (which is the command used to change file permissions). Frequently though, this results in a list which is either too long or doesn't contain what you want (or sometimes both).

### 5.3 Man page sections

The man pages are divided into different sections. Here's a brief description of each (more detailed descriptions later):

- Section 1 User commands
- Section 2 System calls (for C programming)
- Section 3 Library calls (for C and Fortran programming)
- Section 4 Devices, special files
- Section 5 File formats
- Section 6 Games
- Section 7 Miscellaneous
- Section 8 Administrative commands

Typically, when referring to a man page, the section is shown in parenthesis after the command, e.g. `chmod(1)`. This is included to help you find the man page, because some things occur in more than one section. For instance, there is a `chmod` C function, as well as a user command. The C function is referred to as `chmod(2)`, since its man page is found in section 2. Note that in some instances the number may be followed by an additional letter indicating a subsection.

Since man pages with the same name can occur in different sections, you may need to specify the section when using the man command. For instance, to get the C function `chmod`, you would need to say "man 2 chmod", otherwise you'd always get the `chmod` in section 1. The man command will search all sections (not in order but usually starting with 1) until it finds a match, so if you wanted the man page for `fork(2)`, you could just say "man fork", since there is no `fork` in section 1.

### 5.3.1 Section 1

This section contains user commands - things you'll be typing on the command line or using in shell scripts.

The SYNOPSIS section tells you how to use the command, in a somewhat cryptic fashion. Optional parameters are displayed in square brackets. An elipsis indicates that a parameter can be repeated several times. For example the `chmod(1)` SYNOPSIS is

```
chmod [ -fR ] mode filename ...
```

Which means that you would type at least "chmod mode filename" but possibly "chmod -f -R mode file1 file2 file3". Man pages don't always follow these conventions exactly, but it's a start.

Frequently man pages in section 1 have an OPTIONS section which describes what each option does.

### 5.3.2 Section 2

This section contains the C functions which are system calls - requests made directly to the kernel.

The SYNOPSIS section shows you any include files that are required to use the call, and shows the declaration of the function. This declaration is how the function itself is declared. Don't confuse this with how you have to declare the variables you pass to the function, as they will frequently be different in the case of pointers.

There are usually RETURN VALUES and ERRORS sections which indicate what the call will return, and what sort of system errors it can encounter.

### 5.3.3 Section 3

This section contains the C and Fortran Library routines; slightly higher level stuff than the system calls. The format is the same as section 2.

```
length < 50
```

14. Print the fields in reverse order:

```
{ for ( j=NF; j>0; j-- ) printf("%s ", $j)
  printf("\n")
}
```

15. Total up all of the fields in a file:

```
{ for ( j=1; j<=NF; j++ ) total += $j }
END { print( total ) }
```

16. Renumber a file:

```
{ $1 = "" ; print NR, $0 }
```

The "\$1 = "" removes the old line number. To renumber by tens:

```
{ $1 = "" ; print 10*NR, $0 }
```

The format of the awk program is:

```
pattern { action }
pattern { action }
pattern { action }
```

This can be repeated many times in an `awk` program.

- If *pattern* is blank the action is done for all lines.
- If *action* is blank the entire line is printed.

`awk` uses `egrep` extended regular expressions.

Two special patterns exist: `BEGIN` and `END`.

The action for `BEGIN` is processed prior to the first line of the input. This can be used for printing headers and setting field and record separators. The action for `END` is processed at the end of the input. This can be used for report summaries.

In a given record, the input fields are referred to as \$1, \$2, \$3 ...

The entire record is referred to as \$0. Awk has several escape sequences like "C". These can be used in strings.

NF is the number of fields. \$NF is the last field in the record.

5. Print the last field of the last line:

```
    { field = $NF }
END  { print field }
```

field is a variable.

6. Print lines with more than 4 fields:

```
NF > 4
```

7. Print the word count for the file:

```
    { words = words + NF }
END  { print words }
```

This could have been done with `wc`

```
wc -w
```

8. Print lines with "Alaska":

```
/Alaska/
```

9. Count the lines with "Alaska":

```
/Alaska/ { count++ }
END      { print count }
```

10. Find the largest third field:

```
BEGIN    { max = -1000000 }
$3 > max { max = $3 }
END      { print max }
```

11. Find the largest third field and save the line, too:

```
BEGIN    { max = -1000000 }
$3 > max { max = $3; savedline = $0 }
END      { print savedline }
```

12. Print every line that has less than 4 fields:

```
NF < 4
```

13. Print every line shorter than 50 characters:

### 5.3.4 Section 4

This section contains the man pages for all of the device drivers on the system.

The CONFIG section(s) contains information on how to compile that device driver into the kernel.

The DIAGNOSTICS section lists the sort of error messages that might be spit out due to problems with the device driver.

### 5.3.5 Section 5

File formats. This is where formats of lots of different files are found, including both system and user files, a.out format, core file format, and lots of others.

### 5.3.6 Section 6

This is where the man pages for the games live. In general, most games we have don't have man pages.

### 5.3.7 Section 7

A small number of miscellaneous things. Check the introduction to section 7 (see below).

### 5.3.8 Section 8

This section contains various system administration programs.

## 5.4 Layout of man pages

There are certain conventions which are usually followed in man pages. One of them discussed above is inclusion of the section number in parentheses for all references to other man pages or commands which have man pages.

Most man pages are divided into sections (don't confuse this with the sections discussed above, and please let me mix my terminology a bit), some of which are pretty standard. They all have a NAME section, which gives the name of the topic along with a very brief description. The DESCRIPTION section contains a detailed description of the topic. There is usually a SYNOPSIS section, which describes how to use the command. The SEE ALSO section tells you what other man pages (and sometimes other references) to check for information. The FILES section lists what files might be related to that command. If you're lucky there will be an EXAMPLES section.

The order of these sections is pretty standard. Here's a list of most commonly occurring sections, in the order they are likely to occur:

- NAME
- SYNOPSIS
- DESCRIPTION
- OPTIONS
- EXAMPLES
- RETURN VALUES
- ERRORS
- FILES
- SEE ALSO
- NOTES
- WARNINGS
- BUGS
- AUTHOR

## 5.5 Section Introduction

Every section of the man pages will also have an introduction. For instance "man 6 intro" gives an introduction to the games section of the man pages. These introductions typically contain a list of the system-shipped man pages in that section,

## 6.16 awk

**awk**, (from the initials of Alfred V. **A**ho, Peter W. **W**einberger, and Brian W. **K**ernighan) is an interpretive programming language for processing text files. Of all 6 possible combinations of the three initials, **awk** seems most appropriate. At times the language seems awkward (to say the least). It is also very powerful for text processing. It has many of the same features as "C".

The AWK language is a good text-processing language. It has more features and capabilities than SED, but less than Perl.

The book: The AWK programming Language [1] is a good awk reference. It was the source for parts of this lab.

Also look at: UNIX System V Release 3 Programmers Guide [2], Chapter 4 AWK.

The program can be placed on the command line or in a file. The program can read and write files named within the program. You can also pass arguments to the program from the command line.

Many "useful" awk programs are only 1 to 4 lines long! It may be one of the easiest ways to rearrange the order of fields in a file or do a summary report. Some examples of short **awk** programs:

1. Print the number of input lines:

```
END { print NR }
```

This could have been done with **wc**

```
wc -l
```

2. Print the 40th line:

```
NR == 40
```

3. Print the lines 40-50:

```
NR >= 40 && NR <= 50
```

4. Print the last field of every line:

```
{ print $NF }
```

to:

This is the input to the sed lab.  
It is in a file in `~cis137/sed.text` on the class system.  
This line has the numbers 234567890 and 1  
This line has the numbers 265485941 and 3  
And this line too.

Turn in this script program marked with:

*your-name*  
McGregor CIS137  
Lab 6.15: sed program

and may include other information. The section 2 introduction, provides a much too brief description of the inner workings of UNIX, together with a list of system calls, and system error codes.

## 6 Labs

### 6.1 Setup UNIX account

Setting up your account on a system is often done by the system administrator, but not always. Often, making a few changes can make an account more usable for the individual. On UNIX systems there are a number of files that affect the way your account behaves. Some of these can be changed by the individual user, others are changed by the system administrator. The files that affect your account that can be changed by you will usually have a period as the first character of the file name. This makes it a “hidden” file that does not show up in most directory listings.

#### 1. Connecting to the system.

You need to establish a connection to the system. In today’s environment there are several ways to connect.

Once you have reached the system, go on to the next step — logging on to the system.

You are welcome to use any UNIX system that you have access to. The labs in this manual were created with gort in mind, it should be possible to complete them on any UNIX system, though slight changes might be necessary.

#### **gort**

- **A Network connected Windows system** in the computer lab (TBA). Use putty with OpenSSH to access gort from the lab. It is on the desktop. Open a connection to `gort.cscwc.pima.edu` or `144.90.66.69`.
- **Web Browser.** Use `telnet://gort.cscwc.pima.edu` as a URL from most web browser programs. The web browser must have a telnet application configured. Several versions of Microsoft telnet have flaws. The use of putty and OpenSSH connections is strongly recommended when connecting from a Microsoft Windows based system.
- **Internet Telnet.** If you have access to an internet-connected shell account, you can also telnet to the system.

```
telnet gort.cscwc.pima.edu
or
telnet 144.90.66.69
```

2. Login to your gort account.

You should have the “login:” message at this point. Your *username* is: first initial, last name — no space between — maximum 8 characters total, all lower case. Enter your *username* followed by a carriage return or enter. The system will prompt you for your password. Your initial *password* is: student services number — no hyphens. Enter your *password* followed by a carriage return or enter. The system will not echo your *password*. If you do not get logged on try again a few times, then see the lab about your account on gort.

3. The system prompt.

The prompt from the system is programmable. In the following examples I will use:

```
gort:~>
```

for the system prompt. “*username*” will be replaced with your username. “gort” is the name of the system. The “~” will be your current working directory. On gort my prompt looks like:

```
hmcgrego@gort ~ $
```

Your actual prompt may have other information and look quite a bit different. If you are using **bash** as your shell, you can change it by editing your *.bashrc* file.

4. Change your password.

Choose an 8 character password with digits, upper case, and lower case characters. Use the **passwd** command. It will prompt you for your current password, and then for you to enter a new password once for the password, and again for confirmation.

```
gort:~> passwd
```

5. Hidden files in your account.

Your account will have several files in it when you log on for the first time. These files all start with a “.”. This is the UNIX version of a hidden file. To “see” these files you will need to do a **ls -al** command. **ls** (list) is the UNIX directory command.

```
/Some/          lines with the string Some
/Start/,/End/  Lines, inclusive, starting with a
                line containing Start and ending
                with a line containing End
```

Commands are one character following an optional address.

```
s/regular expression/replacement/flags
```

If the *replacement* contains an “&” it represents the matched *regular expression*. *flags*  
**g** — global; **p** — print after replacement; **wfile** - write to file *wfile* if replacement was made.

```
p      Print the current "pattern space".
```

```
q      Quit processing.
```

```
d      Delete the pattern space. Do not print.
```

```
a\  
text  Append text to output prior to reading next line.
```

```
i\  
text  Insert text to output.
```

```
y/abcd/ABCD/  
      Transform any matching character in the  
      first string to the corresponding  
      character in the second string. Strings  
      must be same length.
```

Write a **sed** script/program that changes:

```
/usr/users/csc137/sed.text from:
```

```
This is the input to the sed lab.
It is in a file in ~cis137/sed.text on system gort
This line has the numbers 123456789 and 0
This line has the numbers 154374830 and 2
And this line too.
```

## 6.15 sed - A Stream Editor

`sed` is a text editor. But, it is NOT an interactive text editor. You need to decide exactly what and how you are going to edit prior to getting started. If you have multiple files that need the same changes it is a good choice. If you only have one file and the changes are complex, find another editor — `sed` is not your answer.

`sed` works by reading one input line into a buffer, referred to as “pattern space”, processing all of the commands related to that line, normally printing the resulting line, then reading the next line

In addition to the standard output, results can be directed to several files. Additional input can be read from files and from the program.

In addition to the “pattern space” there is a “hold space”.

Invoking `sed`:

```
sed [-n] [-e script] [-f sfile] [file...]
```

`-n` is used to suppress the printing of lines after processing, normally used in conjunction with the `p` (print) command.

`-e` is used to indicate that the following argument is the program. This is normally optional.

`-f` is used to indicate the file that contains the text processing program.

The input to `sed`, by default, comes from the standard input. If a file name(s) is provided the input comes from the file(s).

Output from `sed` goes to the standard output. It does not modify the input file(s).

Addresses and Commands:

```
[address [, address] ] command [arguments]
```

An address is either a line number, a “\$” (the last line), or a context address, “/regular expression/”, in the style of `ed`.

```
1,$      All lines
32       Line 32
```

```
ls -al
```

- Print out your line of the password file. Please note the space after your name and before the file name `/etc/passwd`. It is needed.

```
gort:~> grep your-username /etc/passwd | lpr
```

- Log off.** This is important. System `gort` has no fixed user license, but it does have limited resources.

```
gort:~> exit
```

Turn in the printout from part 6 marked as follows:

```
your-name
Harry McGregor CIS137
Lab 6.1: Setup UNIX account
```

Labs to be completed in class, otherwise printed labs are due at the end of the week, turned in during class..

## 6.2 Some UNIX commands

The purpose of this lab is to get some familiarity with the UNIX user interface. Please take some time with this and explore. What you learn in this lab will probably help you the entire semester, if not longer.

Commands typed in at a terminal or in a shell script (a batch file) are first interpreted by the shell. The results of the shell processing are used to start executable programs. Executable programs must be marked in the directory as executable, be on the search path, and have the right “magic number” in the first two bytes. Almost all UNIX commands are “external” commands. External commands are commands that are not integrated into the shell, but instead are individual executables.

This lab is intended to introduce some UNIX commands and shell syntax. The person-machine interface is straightforward character-based. The system issues a prompt, to which the user provides a reply. This reply is in the form of the name of an executable program, arguments and options to be passed to the program, and ends with a carriage return. After the carriage return, the shell “looks” at the text string and tries to interpret the request and execute the program. To separate

tokens on the command line use a space (or a tab). Most options are indicated by a preceding hyphen “-”. Unlike DOS, you need a space prior to the options. For example, in DOS you can use “cd .”, but in UNIX you need “cd ..”. (It is hard to see, but there is a space between the “cd” and the “..”.) Be careful about uppercase characters. Almost all UNIX commands are in lower case.

Please look at table 6.4 for some initial UNIX commands (It is a partial list!).

1. Log on to the system.
2. At the system prompt, (in this document it is shown as `gort:~>` – even though it may look **VERY** different), issue the `cal` command. This will show the current month’s calendar.

```
gort:~> cal
```

Try the same command with some arguments. For October 1996

```
gort:~> cal 10 1996
```

Try other dates as well.

```
gort:~> cal 1996
```

McGregor CIS137

Name:

The first “trick” question. Try the command:

```
gort:~> cal 10
```

What year did you get? (A second hint: The year is on the top line.) `cal` only handles a.d. years.

3. Create a file using redirection. Unix lets you direct the output of a program to a file with the “>” symbol. Do this.

```
gort:~> cal 1752 > temp
```

To make the file a bit longer let’s add year 1753.

McGregor CIS137

Lab 6.14: Exercise for `bc`:

**your name:**

1. What are the first 4 digits and last 4 digits of  $512^{512}$ ?

How long did it take to compute on your machine? This can be used as a short and simple benchmark for UNIX systems.

What computer did you use? (If not gort)

2. What happens if you use an `obase` of 100?

3. If scale is set to 75, what are the last 5 digits of the sine of 1.0 radian  
.8414709848.....?????

4. Convert 4BC4A base 13 to base 5. (Make sure that you set the output base first. If not, the new input base will be used when inputting the value for the output base.)

**Turn in this page.**

scale=50

(This needs to be less than 100 on some systems.)

```
gort:~> cal 1753 >> temp
```

Use the years 1752 and 1753; this will be important later. The “>>” will append the output of the command to the file. Another option, not used here, is “>!” which will overwrite a file.

This should quickly come back with the system prompt as if nothing happened. Look at the file *temp* with the `cat` command. (The UNIX “`type`” command. It is short for catenate).

```
gort:~> cat temp
```

It should scroll off of the top of the screen. Look at the same file with the `more` command.

```
gort:~> more temp
```

At the “`--More--(nn%)`” prompt try a space first. Re-issue the command. You can use the cursor control keys to retrieve the last command and edit it in most modern shells. Back in the `more` command try a `cr` (carriage return). Try a “`4`” then a return and then a space. Try an “`h`” for help. A “`b`” for back and a “`q`” for quit. What was the first line output after a “`b`” command (be quick – it scrolls off of the screen!)?

If your system has a `less` command, try it. `gort` has it. The command has several nice features that `more` is missing. This includes backing up even when the input is a pipe.

How many days did September of 1752 have? This may be a **trick** question. This is the year England started using the Gregorian calendar. You really should notice some days missing. (If not, look again!)

You may want to look at the `date` command too. UNIX systems can deal with multiple time zones (for different users on the same system), and leap seconds.

4. The UNIX `ls` command is every other system’s `dir` command. Try it.

```
gort:~> ls
```

Please note that UNIX systems can be customized quite a bit. Your system administrator may have “enhanced” & and “improved” your interface in a

variety of ways! You may be able to use the command `dir`. The command `ls` may have some of its options already set for you. To find out (most) of this run the

`alias`

command. This will give you a list of “command translations” that are taking place. If you find an alias for `ls` that you would like to have “go away” so you can see the “normal” operation you can remove the alias with:

`unalias ls`

You should at least see your file `temp`. Try it with a `-a` option.

```
gort:~> ls -a
```

This will show your hidden files — the ones placed in your account when it was set up. Try the long format, the `-l` option. Also try both the “`l`” and the “`a`” option at the same time ( `-al` ).

```
gort:~> ls -al
```

This displays quite a bit of information about the file. It includes the protection bits, owner, the group (on some systems this will be missing), length in bytes, time of modification, and the file name. What is the length of the file `temp` in bytes? This is the number just prior to the date.

5. Try the `man` command. (Most systems might call it `help`.) It accesses the manual pages, which are cryptic. Look up the manual pages for the `man` command:

```
gort:~> man man
```

Also look up the `ls` command.

```
gort:~> man ls
```

What does the `-R` option do? (Note that the “`R`” is upper case.)

1. No semicolon is needed at the end of a line.
2. Variable names are only a single character.
3. The `define` statement is used to define functions.

Other features of the `bc` program are:

- Control statements are `if`, `while`, `for`, `break`, and `quit`.
- `quit` returns to the shell.
- Comments are `/* ..... */`
- Arrays use square brackets `[ ]` for indexes.

`bc`, when invoked with the `-l` option, has a limited math library. When this is used you have 6 less variable names.

- `s(x)` — sine function
- `c(x)` — cosine function
- `e(x)` — exponential
- `j(n,x)` — Bessel function
- `l(x)` — log
- `a(x)` — arctangent

It also deals with multiple bases. The input base is stored in `ibase`. Be cautious setting conversion bases, `bc` inputs ALL numbers using `ibase`. The output base is stored in `obase`. To convert 4521 from base 7 to base 13:

```
obase = 13
ibase = 7
4521
result returned
```

The fractional precision of the calculation is maintained in “`scale`”. For example:

2. Make sure the directory permission bits allow both world read and execute on the directory.
3. Create the file `index.html`. This file needs to have HTML in it. The HTML needs to have at least the following features:
  - `<html>` and `</html>` tags
  - `<head>` and `</head>` tags
  - `<body>` and `</body>` tags
  - `<title>` and `</title>` tags
  - At least three links to other pages using `<a ...>` and `</a>` tags.

4. Make sure the permission bits for `index.html` are at least world read.
5. Access your page from a web browser and print it off on a printer. The URL to access your page will be:

`http://gort.cscwc.pima.edu/~user-name`

or

`http://gort.cscwc.pima.edu/~user-name/index.html`

Turn in the source html and your printed results marked with:

`your-name`  
 McGregor CIS137  
 Lab 6.13: HTML lab

### 6.14 bc - Extended Precision Calculator

`bc` can compute to at least 99 places right of the decimal point and at least 4000 to the left. This may not be fast and it may not be useful! But, it can be done. `bc` uses `dc`. `dc` uses reverse polish notation. It can be used from the keyboard. Most people will probably prefer `bc`.

If an expression does not include an assignment, the results are printed.

`bc` is a language as well as a desk-top calculator. The syntax is very much like “C”. With the following (and more) exceptions:

What does the `-r` option do? (Note that the “r” is lower case.)

6. Try the `who` command.

```
gort:~> who
```

Also try it with the “`am i`” options (no dash this time).

```
gort:~> who am i
```

Also try the `finger` and `w` commands.

7. Other commands to look up in the text book and the `man` pages: `id`, `date`, `pwd`, `wc`, and `tty`. Please look at the manual pages! Try some of the other options available on the commands, such as:

```
gort:~> ls -iF
```

Two other commands of interest are `chfg` and `chsh`. `chfg` is disabled on some systems.

8. For the more daring: run a shell from within a shell. `tcsh` is an enhanced “C” shell. You will get a prompt ending with “`>`”. You will need to do a `^D` or an `exit` to get back to the original shell. The `^D` is the systems End-Of-File (EOF) character. Some shells disable the use of `^D` to exit the session. You can also try the c-shell. Its name is `csh`. We will mostly use the `bash` shell this semester.

```
gort:~> tcsh
```

9. Log off

```
gort:~> exit
```

This will keep someone else from using your account and free up system resources.

Turn in the pages with your answers (or a copy). Labs to be completed in class, otherwise printed labs are due at the end of the week, turned in during class..

### 6.3 Editor 1 – About Me

Text editing is a required tool for computer users. It is worth spending the time to select a text editor that will help you get the most effective use of your time. All UNIX systems have **ed** (or **ex**) and **vi** text editors. Many will have **emacs**, **pico**, and **xedit**. All text editors have their place. Some are better for specific applications than others. Some require more resources to run. **ed** requires very little. **emacs** and **xedit** require the most. In this lab use **vi**, **pico**, **xedit** or any other UNIX editor. **pico** and **xedit** each have (for me) a simple and easy-to-learn user interface. These editors are the easiest to use. **ex** (or **ed**) are line oriented text editors that I find difficult to use.

- **ed** is a line-oriented text editor. It looks a lot like **ex**, another line oriented text editor. It is the basis for the **vi** text editor. It can be used from any terminal. (Do you still use a KSR-33?) If you need to set up a system or an account but cannot get a full screen editor to work, you may need to use **ed**. However, it is likely that just issuing one command (**setenv TERM vt102**) can get a full-screen editor to work. For the most part, you will want to use a full-screen text editor such as **vi**, **pico**, **xedit**, or **emacs**. You can use all of the **ed** commands in **vi**, so it can be a good place to start.
- **vi** is one of the first full-screen text editors. I have been told that the author of **vi** now uses **emacs**. **emacs** and **pico** may not be on your system though. So initially you may need to use **vi** until you can locate an easier-to-use full screen editor. **vi** has various modes. Which mode you are in does not always show on the screen. It can be confusing to use. The system administration command **vipw**, used to edit the password file */etc/passwd*, initially uses the **vi** text editor. **elvis** is a “clone” of **vi**.
- **pico** is a small, simple, easy-to-use, text editor. It does not have a lot of fancy features. It is always in insert mode. There are two lines of help information at the bottom of the screen. It does not handle large (multiple megabyte) files very well. It is available off of the Internet for free. Installing **pico** is fairly easy.
- **emacs** is a very capable text editor, and very large. On the gort system the **emacs** executable (1.8 Mbytes) is more than 14 times as large as the **pico** executable (.13 Mbytes). **emacs** does a lot of things very well. This includes the ability to control certain models of Coffee makers. The full (free) distribution takes about 40 Mbytes on most systems. For documentation there are books, a tutorial, and an on-line (in **emacs**) hypertext manual.

```
ftp oak.oakland.edu
/pub/simtelnet/msdos/x_10/00_index.txt
```

```
ftp boombox.micro.umn.edu
/pub/pc/packet-drivers/notes/read.me
```

Login as **anonymous**. Then for your password enter your e-mail address. (*username@gort.cscwc.pima.edu*) Once you are in, many UNIX commands work: **cd**, **ls -l**, but **not cat**. You also have the command **ascii**, for use prior to getting ASCII files, and the command **binary** for use prior to transferring binary files. You may also want to try the commands **help**, **get**, **mget**, **prompt**, and **quit**. Remember that this gets the files to gort, NOT to your home PC.

Print a copy of your short file to turn in.

Label your output with:

```
Your-Name
Lab 6.12: Internet
McGregor CIS137
the call number of Living within Limits
the earthquake information
```

Labs to be completed in class, otherwise printed labs are due at the end of the week, turned in during class..

### 6.13 Text processing - HTML

HTML (Hypertext Mark-up Language), through the WEB, has become in the last few years, the primary way of distributing information on UNIX-based systems. A March 1997 survey indicated that about 70% of the servers in the WWW (World Wide Web) are running on UNIX systems.

For this lab, I would like you to:

1. Create a subdirectory in your area called **public.html**.

## 6.12 Internet

The Internet is now connected to most systems. It is a useful tool for locating information and transferring files. Most usage of the Internet today involves the World Wide Web (WWW). This lab looks at accessing other services.

1. **telnet** is a remote terminal protocol and program to access other Internet connected computers running a telnet daemon. Try the command:

```
telnet sabio.arizona.edu
```

to access the UofA library system. Use terminal type “vt100” and respond with a “V”

What is the call number of the book: Living Within Limits [19]?

2. Try out the **finger** command. This sends a short message to a system on the network to see if a user is logged on, and how active. If no user name is given it may show you who is currently on the system. Some systems disable the **finger** command. It is considered a security problem by many system administrators.

Try the following:

```
finger hmcgregor@osef.org
```

and

```
finger quake@gldfs.cr.usgs.gov
```

Write down only the last “large” ( >5 ) earthquake’s time, latitude, longitude and location.

3. **File Transfer Protocol**, (**ftp**) is used to transfer files from one system to another. There are many (>10,000) sites that you can **ftp** to as an anonymous user. I would like you to **ftp** a text file and print it.

These sites and files are suggestions. Any ASCII text file of reasonable length will do. (In other words — keep the file less than one page)

When prompted for a username use “ftp”. For the password use your e-mail address. (*username@gort.cscwc.pima.edu*)

```
ftp sunsite.unc.edu
/pub/Linux/README
```

- **xedit** is a “simple” text editor. It requires an X-11 system (from MIT) to run. So, the text editor may be simple, but the support needed is substantial. At Pima Community College we currently do not have X terminals available for students.

Please do the following:

1. Log on to a UNIX system.
2. Create, using the UNIX editor of your choice, a file with the following information about yourself. Place each of the first 4 items on separate lines. After the first 4 items, skip a line and enter a 5 to 10 line paragraph about yourself. Include your interests and why you are taking this class. Use the file name *about.me*

```
Your-full-name
day and month of your birthday NO year needed
My favorite color is color
My least favorite color is color
```

```
A paragraph - 5 to 10 lines about
yourself and/or your interests.
```

3. Print out the file *about.me*

```
gort:~> lpr about.me
```

4. Log off

```
gort:~> exit
```

Turn in the printout from step 3 marked as follows:

```
Your-name
McGregor CIS137
Lab 6.3: Editor #1 - About Me
```

Labs to be completed in class, otherwise printed labs are due at the end of the week, turned in during class..

## 6.4 First Shell Script

This lab introduces shell scripts, file protection bits, several commands, the `pico` (or `vi`) text editor, and the back-quote (or grave accent (or back-tick)).

1. Create the following shell script file called `cmd` using the `pico` (or `vi`) text editor:

```
#!/bin/bash
echo your-name
echo McGregor CIS137
echo Lab 6.4: UNIX shell script
echo -n "pwd      >> "
pwd
echo -n "who am i  >> "
who am i
echo "id          >> "  'id'
echo '$HOME      >> ' '$HOME
```

To do this use the command:

```
gort:~> pico cmd
```

Please note that the “id” is surrounded by grave accents, not single quotes. Also note that the string:

```
$HOME      >>
```

is quoted by single quotes, not grave accents.

2. When you have finished, verify your work with the following command:

```
gort:~> cat cmd
```

This command, `cat`, is short for catenate. It is the UNIX “`type`” command. It can also be used to concatenate files together. The output you get from this command should match the file shown above.

3. Try to run your shell script. This should fail. It might fail in a different way. The example below assumes that you are running `bash`. You might just get `permission denied` if you are running `csh`.

```
gort:~> cmd
bash: ./cmd: Permission denied
```

- (`compile`) Compile a “C” program. Do a second keyboard read to get the name of the file to compile. Save this name for future compiles so if your script receives a null input it can compile the “current” file. You may want to consider handling file names both with and without the “.c” ending.
  - (`run`) Run `a.out` the last “C” program compiled.
  - (`exit`) Exit the script.
  - If any other input is received it should be passed to the shell as given.
- After processing one input line request a new input. Use the “simple commands” case and read.
  - Test all of the options for your shell script.
  - Can you run your script within your script? Write a **yes** or a **no** on the printout you turn in for each shell.

Make a printout of your menu program’s output. To do this use the `script` command.

```
gort:~> script
Script started, file is typescript
gort:~> menuscript
....
gort:~> exit
exit
Script done, file is typescript
gort:~> lpr typescript
```

`script` makes a copy of the standard input to a the file `typescript` so that it can be printed or saved.

Mark the print out your run results and a copy of your shell script with:

```
your name
McGregor CIS137
Lab 6.11: Menu Program
```

Remember to answer the question about if you could run your shell script within your shell script. Labs to be completed in class, otherwise printed labs are due at the end of the week, turned in during class..

```
argc = 4   File name = a.out
arg[1] = A B
arg[2] = c d
arg[3] = e
```

```
argc = 2   File name = a.out
arg[1] = A B c d e
```

```
argc = 2   File name = a.out
arg[1] = $@
```

```
argc = 2   File name = a.out
arg[1] = $*
```

Turn in a copy of this output marked with:

```
your name
McGregor CIS137
Lab 6.10: C Programming
and the results of a umask of zero
```

Labs to be completed in class, otherwise printed labs are due at the end of the week, turned in during class..

## 6.11 Menu shell script

- Use a text editor to create a menu shell script. The shell script should have 8 options, as listed. Do this using the Bourne shell `sh` or the `bash` shell. If you wish you may do one of the scripts in the korn shell, `ksh` or the `tcsh` shell.
  - (1) Do a `ls` for the current directory with the options last specified with `loption`.
  - (`loption`) Set the options for the above `ls` command.
  - (`cmd`) Run the shell script `cmd` from a previous lab. See Lab 6.4: Lab Editor 2 — `pico` (or `vi`).
  - (`month`) Show the current month’s calendar. Use the `cal` command without any arguments for this part.

4. Check the file protection bits. Use the `ls` command. This is the UNIX “`dir`” command. The `-l` is an option for the “long” format output.

```
gort:~> ls -l cmd
-rw-r--r-- 1 hmcgrego 167 Jan 16 10:21 cmd
```

This is showing that I, the user, have read and write access. My group has read access. Everyone else also has read access. No one has eXecute permission.

5. Change the protection bits. To give everyone eXecute permission, use the `chmod` command:

```
gort:~> chmod a+x cmd
```

The `chmod` command also takes octal arguments, for example:

```
gort:~> chmod 755 cmd
```

symbolic names can be easier to remember.

```
r  read
w  write
x  eXecute
u  user
g  group
o  other
a  all users (user, group, and other)
+  add permission
-  remove permission
=  set permission
```

6. Check the protection bits.

```
gort:~> ls -l cmd
-rwxr-xr-x 1 hmcgrego 167 Jan 16 10:21 cmd
```

Note the new `x`'s in the permission bits.

7. Run your shell script and check your output.

```
gort:~> cmd
```

```
Harry McGregor
McGregor CIS137
Lab 6.4: UNIX shell script
```

```
pwd      >> /home/hmcgrego
who am i >> gort!hmcgrego tty04 Jan 16 10:18
id       >> uid=69(hmcgrego) gid=10(staff)
$HOME   >> /home/hmcgrego
```

```
gort:~>
```

You may need to edit the shell script and try again. If you do, you will not need to set the eXecute bits a second time. Once is enough. UNIX keeps the existing protection bits. The location of the file will not change so a `rehash` is not necessary.

8. Print out a copy of your results (when your output is correct). There are two ways to print out your results:

- (a) Pipe your `cmd`'s output into the `print` program. To do this, use the pipe symbol `|`. The command will look like:

```
gort:~> cmd | lpr
```

- (b) Create a temporary file by using re-direction. Then print the file. The output redirection symbol is `>`. When used with an exclamation point, it can be used to over-write an existing file. (`>!`). The commands will look like this:

```
gort:~> cmd >! temp
gort:~> lpr temp
```

The `print` command may be `lp` on your system. To look at the printer queue use the `lpq` command.

Turn in your output from step 8 marked as follows:

```
your-name
Lab 6.4: First Shell Script
McGregor CIS137
```

Labs to be completed in class, otherwise printed labs are due at the end of the week, turned in during class..

```
printf("argc = %d  File name = %s\n", argc, argv[0]);
for ( i=1; i<argc; i++)
    printf(" arg[%d] = %s\n" ,i,argv[i]);
printf("\n");
}
```

Compile and test the program.

- Create the following script file `test1`:

```
#!/bin/bash
#
# L Taber  Argument passing
# test script
a.out $@
a.out $*
a.out "$@"
a.out "$*"
a.out '$@'
a.out '$*'
```

- Test the script file.

```
gort:~> test1 "A B" 'c d' e
```

The output should look as follows:

```
argc = 6  File name = a.out
arg[1] = A
arg[2] = B
arg[3] = c
arg[4] = d
arg[5] = e
```

```
argc = 6  File name = a.out
arg[1] = A
arg[2] = B
arg[3] = c
arg[4] = d
arg[5] = e
```

```
gort:~> a.out (run the program)
```

```
hello, world (resulting output)
```

- Now take a look at the effect of `umask` on file protection bits during file creation.

You may have noted above that the protection bits that you saw on your `a.out` file did not match my example. This is because you may have a different `umask` setting. To look at your current setting try:

```
gort:~> umask
```

Your output should be an octal number that is the current setting for `umask`. If a bit is set in your `umask` it will prevent that bit from being set when a file is created. Set your `umask` to zero.

```
gort:~> umask 0
```

Recompile your program and look at the protection bits. It should now match my example. Now try a more restrictive setting:

```
gort:~> umask 077
```

Recompile your program and look at the protection bits for `a.out` again. This will only allow you to read, write, or execute the program. Your results should look like this:

```
gort:~> ls -l a.out
-rwx----- 1 ltaber 15165 Oct 13 14:30 a.out
```

Try a `umask` setting of `0777`. Write on your lab what happens and explain why.

- Create the file `args.c` with the following C program:

```
/* Program to print arguments to a program */
/* L Taber October 13, 1992 PCC */

#include <stdio.h>

main(argc, argv )
int argc;
char * argv[ ];

{
int i;
```

Table 1: Several UNIX commands

Command	Usage & Function
<code>cal</code>	Displays the current calendar.
<code>cat</code>	Types a file to the terminal.
<code>date</code>	Shows current date and time.
<code>echo</code>	Echoes whatever is on the line.
<code>lpr</code>	Prints a file on the line printer.
<code>ls</code>	Lists a directory.
<code>ls -l</code>	Lists a directory with permission bits.
<code>id</code>	Shows your user id number and name, and your group id number(s) and name(s).
<code>more</code>	A paging program.
<code>pico</code>	A simple to use text editor.
<code>pwd</code>	Shows the path to the current working directory.
<code>tty</code>	Shows the communications port you are connected to.
<code>vi</code>	The <b>standard</b> UNIX text editor.
<code>w</code>	Shows who is using the system.
<code>who</code>	Also shows the users that are logged on to the system.
<code>who am i</code>	Displays information about you.

Table 2: Special shell symbols

Symbol	Meaning
<code>&gt;</code>	Send output to a file
<code> </code>	Send output to another program
<code>&lt;</code>	Get input from a file
<code>&gt;&gt;</code>	Append output to an existing file
<code>&amp;</code>	Run command in the background
<code>;</code>	Used to separate two commands
<code>  </code>	Run second command if first fails
<code>&amp;&amp;</code>	Run second command if first succeeds
<code>&gt;!</code>	Overwrite existing output file

Table 3: UNIX String substitution

Grave accents or Back-ticks or Back-quotes	– Near the character “!” & “!” on most keyboards. Causes the enclosed string to be passed to a sub-shell. The result is passed back to the original shell with all new line characters removed. These can be nested with the proper use of escape characters.
environment variables	When referenced in a shell script they need to be preceded with the “\$” character. When the variable is set no, “\$” is used.
single quotes	Prevents the shell from doing any substitution of the string. It is passed directly to the command.
double quotes	Allows substitution of variables prior to passing the arguments to a command. Leaves spaces alone and passes it as a single argument.

## 6.5 Editor – vi

In this lab we will look at some of the features of the vi text editor. This can be a very good text editor for touch typists. The only special key that is needed is the “escape” key. For a full description of the vi text editor look at the manual pages.

1. Copy the file `~cis137/flatpara.text`

```
gort:~> cp ~cis137/flatpara.text .
```

(Did you notice the trailing period? You need it.)

The file is a paragraph from the book *Flatland* by Edwin A. Abbott.

2. Bring up the file in vi.

```
gort:~> vi flatpara.text
```

3. Insert *your name*, the *current date*, and *McGregor CIS137* as the first line of the file.
4. Number each line in the file. Number the line with your name as 1. Place a space after each number. You should have 19 numbered lines. This needs

Labs to be completed in class, otherwise printed labs are due at the end of the week, turned in during class..

## 6.10 “C” Programming Language

Most of the UNIX operating system is written in the “C” programming language. Most of the tools (and games) that run on the system are also written in “C”. Many other utilities have a flavor of “C”, such as the “C” shell `cs`, `awk`, and `perl`. The interface to the operating system is easily accessed with the “C” language. The operating system calls look like “C” function calls. The system calls are documented in section 2 of the manual pages. Library calls are documented in section 3 of the manual pages.

This lab will also be used to look at the effects of `umask` on protection bits when a file is created.

- Using a text editor, create the file `hello.c` with the following “C” program:

```
/* Program to print "hello, world" */
/* Kernighan & Richie "The C Programming Language" */
/* Copyright 1978 page 6 */
```

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

```
main()
{
    printf("hello, world\n");
}
```

- Compile and run the program:

```
gort:~> cc hello.c (cc is the standard C Compiler)
or
gort:~> gcc hello.c (gcc is the gnu C Compiler)
```

The standard output of the compiler is *a.out*. You can use the option `-o` to put the executable in another file.

```
gort:~> ls -l a.out
-rwxrwxrwx  1 ltaber  15165 Oct 13 14:30 a.out
```

## 6.9 Three bash scripts

Using a text editor, create three (3) **bash** shell programs. You have many options. This is a list of possibilities. If you have a specific project in mind, check with me.

- Write a prime number program using the **bash** shell. Accept an argument and print out all prime numbers less than the passed argument. (Look at `factor(1)`.)
- Get current earthquake information from the Internet. Look at the quake alias for starters. (The Internet keeps changing.) Strip out all of the data except the large earthquakes (`grep`). Place this information in a file (create it if necessary). Then take this file and sort (`sort`) it. After it is sorted remove lines that are the same (`uniq`). Display this result and place it back into the above file for next time. This should work in such a manner that if you run it at a later date, the file accumulates earthquake information. The shell script needs to store information from one login to the next.
- Process the `/etc/passwd` file to create a list of unique user numbers and a list of unique group numbers. Title each list.

- List the last (about) 25 users that are on (were on) the system in alphabetic order by user name. Use the

```
last -100
```

command to create a list of possible users. Then use `cut`, `sort`, and `uniq` to arrange the list.

- Write a “tree” script. Do a recursive directory search and display it with an indented format. There is no need to attempt to use special graphic characters. As the first and only required argument, pass the path location to start your tree. Do not display the files in the directories, unless that is an option of your shell script.
- Write a “useful” shell script, and a note to me explaining why it is useful.

Make a printout of your output and shell script, mark it with:

```
your name  
McGregor CIS137  
Lab 6.9: Three bash scripts
```

to be placed into the file by hand. You may want to use the command `:set number` to check your numbers. In some versions of `vi` you will need to get out of insert mode with the `esc` key to move from one line to the next.

5. With the “:” command move to line 11. `:11<cr>`.
6. Move to the end of the line with the “\$” in command mode. Delete the “;” with the `x` command. Move to the beginning of the line with a “0” (zero). Replace the “1” with a “4” using the “r” command. If you are on an X system, try positioning the cursor with the mouse.
7. On lines 15 through the end of the file change all of the “t”s to “\*\*”. Use the “:” command “:15,\$s/t/\*\*/g”. In this command “t” is a regular expression.
8. Read the manual pages of `ed` or `ex`. If `ed` is not on your system try `ex`. This is where regular expressions may be documented. Read the section on regular expressions. On gort `man` uses `less` to help the user view the manual pages. `less` uses regular expressions for searching, like `vi`. So the “/” and “?” work for forward and backward searches. You may need to look at the `vi` manual pages. You can use regular expressions in many UNIX commands. We will come back to regular expressions in several future labs.
9. Go to line 4 of the file with a “:4”. Forward search for “land” with the “/” command “/land”. Repeat the search twice with two “n” commands. To line four (4), the blank line, add the line number that the cursor is on. The “?” is used to search backwards. An “N” searches again, but in the other direction.
10. Replace on lines 5 through 10 all single letters between two vowels with
  - (a) A pound sign “#”
  - (b) The first vowel
  - (c) The letter in the center
  - (d) A “\$”
  - (e) A second copy of the letter
  - (f) the second vowel and
  - (g) A second “#!”

Now for the command to do this.

```
:5,10s/\([aeiou]\)\([a-z]\)\([aeiou]\)/#\1\2$\2\3#/g
```

This needs an explanation. The “5,10s” indicates what area of the file. Between the first two forward slashes “/” are three regular expressions. They are marked off with “\” and “\”. The first and last specify vowels, [aeiou]. The middle one, [a-z] is for all lower case letters. These regular expressions are referred to later by “\1”, “\2”, and “\3”. The “g” at the end is for global, all of the occurrences on a line.

11. Exit the vi editor with the command “ZZ” (Upper case). This will save your file.
12. Print out your resulting file.

```
gort:~> lpr flatpara.text
```

Turn in your output from step 12 marked as follows:

```
your-name
Lab 6.5: vi Part 2
McGregor CIS137
```

Labs to be completed in class, otherwise printed labs are due at the end of the week, turned in during class..

```
lrwxr-xr-x 1 root      7 Feb 24 1993 lib -> usr/lib
drwxr-xr-x 2 root    8192 Feb 24 1992 lost+found
drwxr-xr-x 2 root     512 Aug  7 1992 mnt
drwxr-xr-x 2 root     512 Feb 24 1993 opr
drwxr-xr-x 3 root     512 Aug  3 12:44 public
lrwxr-xr-x 1 root      7 Feb 24 1993 sys -> usr/sys
drwxrwxrwt 5 root    1024 Sep 12 18:46 tmp
-rw-r--r-- 1 root   11568 Jul 24 1992 ultrixboot
drwxr-xr-x 27 root    1024 Jun 10 11:28 usr
drwxr-xr-x  9 root     512 Aug 19 15:43 usr2
-rwxr-xr-x  1 root  3448280 Feb 24 1993 vmunix
```

This is what I would like your output to look like:

```
September 1993 has 30 days
The root directory on gort has 12 directories
```

```
2048 r-x  2 14:26 bin
 512 r-x  2 15:59 bsd-src
3072 r-x  3 14:45 dev
6656 r-x  6 22:25 etc
8192 r-x  2 1992 lost+found
 512 r-x  2 1992 mnt
 512 r-x  2 1993 opr
 512 r-x  3 12:44 public
1024 rwt  5 18:46 tmp
1024 r-x 27 11:28 usr
 512 r-x  9 15:43 usr2
```

Make a printout of your output and shell script, mark it with:

```
your name
McGregor CIS137
Lab 6.8: Shell script --- dir
```

Labs to be completed in class, otherwise printed labs are due at the end of the week, turned in during class..

### 6.7.3 extended regular expression rules

See the `egrep(1)` manual pages for a complete description. `egrep` can run up to 10 times faster than `grep`. Its memory usage is less predictable. `awk` also uses extended regular expressions.

- All regular expression rules.
- `+` Matches 1 or more of the preceding regular expression.
- `?` Matches 0 or 1 of the preceding regular expression.
- `|` Between two regular expressions `|` will match if either expression matches.
- `( )` Expressions may be enclosed in parentheses for grouping.

### 6.8 Shell script — `dir`

Using a text editor, create a shell program. Your shell program will probably need to use `cut`, `ls`, `expr`, `grep`, `paste`, `wc`, `cat`, `hostname`, and `cal`. It will probably also use pipes, redirection, variables, and command substitution.

Your program should do the following:

1. “Display” the number of days in the current month. For example:  

```
September 1996 has 30 days
```
2. “Display” the number of directories in the root directory. For example:  

```
The root directory on gort has 12 directories
```
3. “Display” the root directory in the following format. This is what the `ls -l /` command gives:

```
drwxr-xr-x 2 root 2048 Jun 30 14:26 bin
drwxr-xr-x 2 root 512 Jun 2 15:59 BSD-src
-rw-r--r-- 1 root 147456 Sep 7 14:25 core
-rw-r--r-- 1 root 925 Apr 8 18:00 dflt.DECterm
drwxr-xr-x 3 root 3072 Sep 8 14:45 dev
drwxr-xr-x 6 root 6656 Sep 9 22:25 etc
```

Table 4: VI Text Editor

Starting	<code>gort:~&gt; vi file</code>	start edit
Stopping	<code>ZZ</code> <code>:wq</code> <code>:q!</code> <code>:w</code>	save edits and exit save edits and exit abandon changes save changes & continue editing
Refresh screen	<code>^L</code>	Clear and redraw screen
Modes	<code>ESC</code> <code>ESC :</code>	forces Command mode forces ed/ex mode
Cursor	use the cursor keys <code>k, j, l, h</code> <code>0</code> <code>^</code> <code>\$</code> <code>M</code> <code>H</code> <code>L</code> <code>^U</code> <code>^D</code> <code>^F</code> <code>^B</code>	if set up. up, down, right, left start of line start of line end of line middle of screen beginning of screen Last line of screen up half a page down half pages forward full page back full page
Text	<code>line numberCR</code> <code>i</code> <code>a</code> <code>r</code> <code>x</code> <code>dd</code> <code>p</code>  <code>P</code> <code>yy</code>	go to specified line insert before append after replace current character delete current character delete line put (insert) last delete or yank put before current character yank to clipboard one line
Search	<code>/textCR</code> <code>?textCR</code>	look forward in text look backwards in text
ed commands	<code>:ed-commandCR</code> <code>:set number</code> <code>:set nonumber</code> <code>:setshow all</code>	ex or ed commands show line numbers do not show line numbers show all set-able items
setup file	<code>.exerc</code>	initialization file

Table 5: vi “:” commands

Command format	In command mode type a “:” A “:” and the cursor will be at the bottom of the screen waiting for a command. First 0, 1, or 2 decimal line addresses or . (current line) or \$ (last line), then a command letter, usually lower case then, optional trailing arguments (and commands). Followed by a carriage return.	
Escaping from	The escape key	
Stopping vi	:w :q :Q	write file exit - no write abandon edits
Delete lines	:d :2d :5,8dp	delete current line delete line 2 delete lines 5-8 and print current line
Un-do	:u :U	un-do last command un-do last line
File commands	:f <i>filename</i> :! <i>shell-command</i>	sets file name Insert shells output
Substitute	:s/ <i>abc/xyz/</i>  :1,\$s/ <i>abc/xyz/g</i>	change, in the current line, the first <i>abc</i> to <i>xyz</i> . change, all “ <i>abc</i> ”s in file to “ <i>xyz</i> ”
move text	:5,8m9	move lines 5-8 to after line 9
copy text	:1,3t8	copy lines 1-3 to after line 8

- \* Matches any string including a null string.
- ? Matches any single character.
- [ ] Matches any enclosed character. A range of characters can be specified with a “-”. [a-d] == [abcd] If the first character following a “[” is a “^” then any character NOT enclosed is matched.
- Periods at the start of file names must be matched explicitly.
- Forward slashes “/” must be matched explicitly.
- If no match is found, no substitution is done. (sh only).
- Matching filenames are alphabetized after a substitution is made.
- A “\” can be used prior to “\*” & “?” to look for “\*” & “?”.
- A list “{i1,i2,...}” expands list - bash, csh, and tcsh.

### 6.7.2 regular expression rules

See the `ed(1)` manual pages for a complete description.

- . Is a one character re (regular expression) that matches any character.
- \* Matches 0 or more of the preceding one character re.
- Inside square brackets “[ ]” matches any enclosed character. A range can be specified with a “-”. [a-d] == [abcd]. If the first character following a “[” is a “^” then any character not enclosed is matched.
- ^ at the beginning of the re forces the re to match at the beginning of a line.
- \$ at the end of the re forces the re to match the final segment of a line.
- A “\” can be used prior to any special character if what is wanted is a match of that character. \ \* ^ \$ [ ] / .

Use the `echo` command to print out *your name* and McGregor CIS137. For each item also provide a short description of what you are printing out.

Start your shell script with `#!/bin/bash` to use the `bash` shell.

1. The line number and the line of all lines that have the string “closegraph”.
2. The number of lines that have the string “include”.
3. The number of lines with a period “.”. (Be careful “.” is a meta character.)
4. The number of lines with a greater than symbol “>”. (Be careful “>” is a shell redirection character.)
5. The line number and the line where the string “main(” is. This function is where a “C” program starts running.
6. The number of lines that have the string “colormap”.
7. The number of lines that have the string “/\*”. (Be careful “\*” is a meta character.)

Watch out for the characters being interpreted by the shell with the use of single quotes. Special regular expression characters need to be Escaped.

Turn in a copy of your shell script and its results. Make a printout of your output and shell script, and mark it with:

```
your name
McGregor CIS137
Lab 6.7: grep & regular expressions
```

Labs to be completed in class, otherwise printed labs are due at the end of the week, turned in during class..

### 6.7.1 sh and csh rules

See the `sh(1)` & `csh(1)` manual pages for a complete description.

## 6.6 Editor 3 – emacs

Do the `emacs` tutorial on a system with `emacs`. `gort` has `emacs` installed. Type `emacs` and follow the instructions.

```
gort:~> emacs
```

then `^H`(for help) then `t`(for tutorial)

You may want to try the hyper-text manual pages for `emacs`.

```
gort:~> emacs
```

then `ESC`, `x`, and `info`

You may want to copy `~hmcgrego/.emacs` from my area so the cursor keys work if you use a VT200/VT300 terminal. This was done for you when your account was setup on `gort`. You may want to ftp this file to another system.

Additional information can be found on `emacs` in the `info` system.

```
gort:~> info emacs
```

and in the O'Reilly & Associates, Inc. book, Learning GNU Emacs [4].

Copy over an `emacs` configuration file from the class area.

```
gort:~> cp ~cis137/.emacs .
```

(Did you see that trailing “dot”? You need it.)

McGregor CIS137 Lab 6.6: `emacs` — Print your name.

1. What are the last 5 words in the `emacs` tutorial?

2. What does the command “gort:~> pico temp” do?

3. What is the -k option for when used with the man command?

4. How can you display just your entry in the /etc/passwd file? (grep)

5. How many lines are in /etc/passwd? (use wc)

6. What are my office hours?

finger hmcgregor@osef.org

7. What does the -F (upper case) option do for the ls command?

8. How many options does the grep command have? (man)

9. What is in the file /var/log/syslog? (cat, tail, head, more)

Write your answers on this page and turn it in.

## 6.7 grep and regular expressions

Regular expressions are used in many UNIX tools. The string substitution used in **grep**, **ex**, and **vi** use regular expressions. The shell, in its file name expansion, uses a simplified regular expression format. **awk** and **egrep** use extended regular expressions.

**grep** is a program used to search a file, or a group of files, for a specific regular expression. It does wonders in finding lost functions and subroutines in a group of program sources, or checking to see if you have removed all references to an obsolete function or variable. An option of **-n** will cause the line number to be printed.

**wc** is a program used to count lines, words, and characters, usually all three. Its options are **-l**, **-w** and **-c**, for lines, words and characters. The default is **-lwc**.

In this lab you will search, using the program **grep**, an old “C” program of written by Louis Taber named **cman6.c**. It is a Mandelbrot set program written for the Borland “C” compiler and some graphics library additions. The file name is **cman6.c**.

Pipes are the use of the “|” character to direct the output of one command into the input of another command. For example:

```
grep 'define' ~cis137/cman6.c | wc -l
```

will extract all lines from **~cis137/cman6.c** that have the string **define** and send them onto **wc**. **wc** will then count the number of lines. **grep** also has a **-c** option that will count the lines that match.

Another interesting option to **grep** is **-v**. This will cause **grep** to invert the condition. If the regular expression matches a line it will **NOT** be output. However, all the lines that didn’t match will be output instead.

The shell, when it is processing a command line, first looks for pipe and redirection symbols. If it finds these it effectively removes these from the command line. The individual commands are unaware that they ever existed. Then it searches for and replaces variables and file names with wild cards. Afterwards it executes the individual commands.

Write a shell script called *greplab* that searches **~cis137/cman6.c** for the various items below.