CS 1355: Introduction to Programming

Unix Guide

with Vi Tutorial

Course Code: 10210CS135501
Quarter: Fall 2013

Professor Pai H. Chou
Computer Science
National Tsing Hua University

September 16, 2013

Contents

1 Login to your Unix account **3**
- 1.1 Application for a CS Account . . . 3
  - 1.1.1 Computer Science students at NTHU . . . 3
  - 1.1.2 Non-CS Majors . . . 3
- 1.2 Software and commands for remote login . . . 4
- 1.3 Unix Shell . . . 5

2 Unix Guide **6**
- 2.1 Getting Started . . . 6
  - 2.1.1 Logging in . . . 6
  - 2.1.2 Typing Commands . . . 7
  - 2.1.3 Mistakes in Typing . . . 8
  - 2.1.4 Strange Terminal Behavior . . . 8
  - 2.1.5 Read-ahead . . . 8
  - 2.1.6 Stopping a Program . . . 8
  - 2.1.7 Logging Out . . . 9
  - 2.1.8 Password Maintenance . . . 9
  - 2.1.9 On-line Manual . . . 10
  - 2.1.10 Files . . . 10
  - 2.1.11 Subdirectories . . . 11
- 2.2 Using the Shell . . . 12
  - 2.2.1 Start-up . . . 12
  - 2.2.2 Filename Completion . . . 12
  - 2.2.3 History Substitution . . . 13
  - 2.2.4 Aliases . . . 13
  - 2.2.5 Using Files Instead of the Terminal . . . 14
  - 2.2.6 Pipes . . . 14
  - 2.2.7 Environment Variables . . . 14
  - 2.2.8 Filenames . . . 15
  - 2.2.9 Job Control . . . 16
  - 2.2.10 Useful Commands . . . 17
- 2.3 Files . . . 18
  - 2.3.1 File Protection . . . 18
  - 2.3.2 Changing Permissions . . . 18
  - 2.3.3 Viewing Files . . . 19

3 The vi text editor **20**
- 3.1 Introduction . . . 20
- 3.2 Entering and exiting the vi editor . . . 20
- 3.3 Modes of Operation . . . 21

3.3.1 Insert Mode . . . 21
3.3.2 Command Mode . . . 21
3.4 Searching for a String . . . 23
3.4.1 Special Characters in Search Strings . . . 23
3.4.2 Search String Special Characters . . . 24
3.5 Issuing ex Commands . . . 24
3.5.1 String Substitution . . . 24
3.6 The Use of Buffers . . . 25
3.6.1 The Yank Command . . . 25
3.6.2 The Put Command . . . 26
3.7 Reading and Writing Files . . . 26
3.7.1 The Read Command . . . 26
3.7.2 The Write Command . . . 26
3.8 Word Abbreviations . . . 26
3.9 VIM-Specific Features . . . 27
3.9.1 Syntax Highlighting . . . 27
3.9.2 Unix, Mac, or DOS File Format . . . 27
3.9.3 Unicode and other encodings . . . 27

4 Setting up SSH Public/Private Keys **29**
4.1 SSH Version 2 . . . 29
4.2 SSH Version 1 . . . 30

5 Typescript **31**

6 Cygwin Installation and Setup (for Windows) **32**

7 Xcode and GCC Installation (for Mac OS X) **36**
7.1 GCC Installation through Xcode . . . 36
7.2 GCC Installation without Xcode . . . 36

Chapter 1

Login to your Unix account

For this class, you will be doing your assignments by *logging on* to a shared machine (server) running the Unix operating system. Even though you may be using a personal computer or a workstation that is capable of computation locally, you will mainly be using them as *terminals* (clients), whose job is to pass keystrokes to the server and display outputs from the server.

To use a shared machine, first you need an *account* on the machine.

The name of the instructional server is cs25.cs.nthu.edu.tw. You can log in to your account with your user name and password. Your account also comes with a certain amount of disk space. You can use this space to store homework assignment files, and you don't need to bring your own disks or other storage media.

If cs25.cs.nthu.edu.tw is down you can try another machine listed upon login. You can use the same user name and password, and your files will be the same.

1.1 Application for a CS Account

1.1.1 Computer Science students at NTHU

OZ account

Please check if you have the password for your oz account. Open your web browser to http://webmail.oz.nthu.edu.tw and see if you can login. If so, then you have an oz account. Otherwise, please go to the NTHU home page http://www.nthu.edu.tw/, click on the link for CC (計算機與通訊中心) at http://www.cc.nthu.edu.tw/bin/home.php click on (申請表下載 > 常用單表), and find the form for 學生電子郵件信箱申請單 to apply for your account.

CS Account Application

After you have your OZ account, you can now apply for the CS account. Go to NTHU's Departmental Computer Center (資工系 系計算機中心) at http://www.cs.nthu.edu.tw/~cscc/ and click on "CS帳號線上申請". Follow the instructions on the page. It may take one working day to get your account created.

1.1.2 Non-CS Majors

Non-CS majors cannot apply for an account online. Instead, the application must be submitted on paper. Please contact one of the TAs for the application procedure.

1.2 Software and commands for remote login

You can connect to cs25.cs.nthu.edu.tw from virtually any computer anywhere that has internet access. What you need is a client program for *remote login*.

Previously, people used `rlogin` or `telnet` to connect to the server, and `ftp` or `rcp` to transfer files. However, they are insecure, because your keystrokes or output are in clear text and can be *snooped* by others. This means your account name and password can be stolen this way. So, for security reasons, do not use either of these programs.

Instead, use `ssh` as the primary way to connect to the server. `ssh` stands for *secure shell*, and it encrypts your network communication, so that your data looks like garbage to snoopers. For file transfers, use `sftp` or `scp`, which are secure. You could also set up an *ssh-tunnel* so that previously unencrypted communications can be encrypted.

Depending on what computer you use, it may have a different *implementation* of `ssh`, but the basic function underneath are all the same. Check out the OIT's page on SSH:
http://www.oit.uci.edu/support/sysadmin/ssh_info.html

- If you are logging in from a Win32 machine (e.g, NT, XP, Vista, etc), you can use `PuTTY`.
- Actually, Win32 users are strongly recommended to install Cygwin, which gives you a unix-like command-line environment. It can be downloaded for free from http://www.cygwin.com/. Once you download the installer (setup.exe), choose the packages for SSH, X Windows, and other tools you find useful. See also Chapter 6 for detailed instructions on how to install Cygwin. If you install the SSH package in Cygwin, then you can just use it instead of PuTTY to log in to the instructional server.
- MacOS X and Gnu/Linux already has this built-in (use Terminal or X11 to run a unix shell). MacOS X users can also consider iTerm, which supports enhanced features. PuTTY is also available on Linux. The command to login to the instructional server is

  ```
  % ssh cs25.cs.nthu.edu.tw -l yourUserName
  ```

  alternatively, newer versions of ssh can use the newer syntax

  ```
  % ssh yourUserName@cs25.cs.nthu.edu.tw
  ```

  It will ask you for the password. However, you can set up your ssh public/private key pairs so that you don't have to type in the password when logged in from your account. See Chapter 4 for details.
- iPhone/iOS users can try iSSH, pTerm, or TouchTerm.
- If you are logging in from an X terminal, you can use the command

  ```
  % ssh cs25.cs.nthu.edu.tw -X -l yourUserName
  ```

  alternatively, newer ssh can support the command line syntax

  ```
  % ssh -X yourUserName@cs25.cs.nthu.edu.tw
  ```

  (note: `%` is the prompt, not part of your command) It will prompt you for your password. Note that the `-X` option allows you to run programs that open X windows on your screen (e.g, Netscape, IDLE, xclock, acroread, mozilla, etc). However, for the purpose of this course, we probably will not make use of any features specific to X-Window.
- If `-X` doesn't work, try `-Y`

1.3 Unix Shell

By now you should be logged in, and you should be looking at the prompt

```
cs25% _
```

Note: in the following writeup, we will show just

```
%
```

for the prompt, instead of

```
cs25%
```

If you login to another machine, such as `malibu`, then the prompt would instead look like
`malibu%`

The fact that `%` is used as the prompt indicates that it is "C shell" (`csh` or `tcsh`), which is Berkeley-style shell and is common for Sun OS environments. Another shell with recent rise in popularity is `bash`, for *Bourne-again shell*. It is written for the GNU free software project and is now the default for MacOS X, Linux, Cygwin, and more. There are some minor differences between `bash` and `csh`, but for the most part, the commands described here are the same.

You should change your password using the `passwd` command. The password will be changed on all the CS Department's Sun machines, not just on cs25.cs.nthu.edu.tw.

Try out the following commands at the shell prompt.

`ls`	list files
`cd`	(change working directory)
`pwd`	(print working directory)
`mkdir`	(make directory)
`mv`	(rename/move files)
`cp`	(copy files)
`rm`	(remove files)
`rmdir`	(remove directory)
`cat`	(print the content of a file)
`more`	(print the content of a file, one screen at a time)
`echo`	(print the arguments on the rest of the command line)

Most commands take one or more file names as parameters. When referring to files, you may need to qualify the file name with directory references, absolute vs. relative paths:

`.`	(current directory)
`..`	(one level higher)
`~`	(home directory)
`/`	the root (top level) directory

Chapter 2

Unix Guide

This chapter is taken from the NACS unix guide, at
http://www.nacs.uci.edu/help/manuals/uci.unix.guide/

It was originally written for c-shell (csh), which is the default shell on SunOS and Berkeley-style Unix. The commands are mostly the same for other shells such as bash (Bourne-again shell), which is now the default shell for Cygwin, MacOS X, Linux, and many recent systems. Their differences are notes in this document.

The following conventions will be used for the text in this manual: The names of commands and programs, and text that you type, will be in "double quotes." Sometimes, when the quotes could be confused with something you're supposed to type, other formats may be used. File names will be in 'single quotes.' Text that you see on the screen will appear in typewriter type.

```
This is an example of typewriter type.
```

Keys on your keyboard are displayed capitalized and in rounded boxes, such as Return and P. Control keys, such as CTRL-D, are typed by holding down the key marked Ctrl, pressing the D key, and then releasing the Ctrl key. (The Ctrl key acts much like a Shift key.)

The chapter "Getting Started" is based on "Beginning Unix" by Brian W. Kernighan, AT&T Bell Laboratories, 1978, with modifications by Cathy Smith, Andy Hall and Steve Franklin.

2.1 Getting Started

2.1.1 Logging in

Before you can log into any of the systems you must have an "account." You can use one of the terminal programs (ssh, putty, etc) mentioned in the previous chapter.

```
% ssh cs25.cs.nthu.edu.tw -l yourUserName
```

If you connect to the server without providing a user name, it will prompt you for it:

```
SunOS UNIX (cs25.cs.nthu.edu.tw)

login:
```

Type in your login name in lower case. (If you type your login name with upper case letters, Unix assumes your terminal cannot generate lower case letters and will translate all subsequent upper case letters to lower case. You do not want this to happen!) After entering your login name, you will be asked for your password:

```
Password:
```

Your password will not be echoed on the terminal as you type it.

If you have entered either your login name or your password incorrectly, a login incorrect message will appear on your terminal, and you will have to repeat the login/password sequence.

When you have entered both your login name and password correctly, you will see a message giving the time of your last login.

```
Sun Microsystems Inc.   SunOS 8
===============================================================
 Host: cs25.cs.nthu.edu.tw              (Sun Blade2000 + 8G RAM)
===============================================================
---------------------------------------------------------------
 All Computer Center information are posted on News: nthu.cs.cc
---------------------------------------------------------------
-----------------------------------------------------------------
| << WARNING >> PLEASE DO _NOT_ submit more than _1_ job  at the  |
|      same time; otherwise, your account will be suspended !!    |
|-----------------------------------------------------------------|
|[警告]為保障使用 CPU 的公平性, 同一使用者, 請勿同一時間執行一個  |
|      以上的 JOB , 違者一經發現, 暫停帳號一週 !! 謝謝您的合作 !! |
-----------------------------------------------------------------


注意:   SUN工作站群的 cs20, cs21, cs22, cs23, cs24, cs25, cs26 等七部
       機器上,限制每位使用者一個機器限一個工作(Job), 違者停用帳號.
       限制每位使用者一個機器限一個工作(Job), 違者停用帳號. 其他
       機器不限制. 請使用者配合. 謝謝.

-- uptime -------------------------------------------------------------
  5:32am  up 200 day(s), 21:42,  1 user,  load average: 0.02, 0.01, 0.01

-- You have 1 process(es) before login [您有 1 個程序尚在執行]----------
   RUSER   PID %CPU    STIME TT      COMMAND
  phchou 11603  0.1 05:32:10 ?       /usr/local/sbin/sshd

>> Please kill unnecessary processes.[請將不用的程序清除]

>> Your mailbox size is [您信箱的大小為] 7211 KB
>> Your mailbox is [您信箱位置為] /user/prof/phchou/Maildir

cs25:~(phchou)%
```

The culmination of your login efforts is a "prompt character," a sign that indicates that the system is ready to accept commands from you. The prompt character is the percent sign, %, unless you've customized it. If you are running bash, then the prompt character is $ by default.

2.1.2 Typing Commands

Once you've seen the prompt character, you can type commands, which are requests that the system do something. Commands are practically always typed in lower-case. Typing "`date`" followed by the Return key will produce something like

```
% date
Mon Sep 16 13:57:55 CST 2013
```

After any system command, you must press Return. The Return key won't be mentioned again, but it has to be there at the end of each line.

If you make a mistake typing the command name, and refer to a non-existent command, you will be told. For example, if you type "daze" you will be told

```
% daze
daze:  Command not found.
```

Of course, if you inadvertently type the name of some other command, it will run, with more or less mysterious results.

Unix commands usually take the form:

- *command -options 'filename'*

where "*-options*" is a list of the options as shown on the manual page, and '*filename*' is the name of the file upon which you want the command to act. The list of options is usually preceded by a "-". You will not have to specify options or file names with all commands. Options are usually single letters which will change the default behavior of a program.

2.1.3 Mistakes in Typing

If you make a typing mistake, and see it before Return has been entered, there are two ways to recover. The Del (Delete) key (sometimes labeled RUBOUT; PC users should use the Backspace key) erases the last character typed; in fact, successive uses of Del erase characters back to the beginning of the line (but not beyond). If you want to erase ALL the characters you have entered on the current line, press CTRL-U (which signifies the simultaneous depression of the Ctrl and U keys) and start the line again. Typing CTRL-W will erase one word at a time.

2.1.4 Strange Terminal Behavior

Sometimes, particularly when you first log in, your terminal may act strangely. For example, each letter may be typed twice, or Return may not cause a line feed or a return to the left margin. This might well be caused by your having designated the wrong terminal type while logging in. The easiest way to remedy the situation is to log out and log back in again. If your terminal is so messed up that you cannot even type a "`logout`", press the Line Feed key, type "`reset`", and press Line Feed again (don't worry if you can see "nothing" on the terminal while you are doing this). If you don't have a Line Feed key on your terminal, type CTRL-J.

2.1.5 Read-ahead

Unix has full read-ahead, which means that you can type as fast as you want, whenever you want, even when some command is typing at you. If you type during output, your input characters will appear intermixed with the output characters, but they will be stored away and interpreted in the correct order. So you can type several commands one after another, without waiting for the first to finish or even begin.

2.1.6 Stopping a Program

You can "abort" most programs by typing CTRL-C. In a few programs, like the vi text editor, CTRL-C stops whatever the program is doing but leaves you in that program.

2.1.7 Logging Out

You log out of a terminal session by typing "`logout`".

If your logout is successful (see below), you will see a message indicating that, a message inviting you to login (again), some garbage or even nothing at all (that is, no familiar %).

There are two "hindrances" to logging out that you may encounter. Both appear as a message after you have typed "`logout`". The first of these is

```
% logout
There are stopped jobs.
% _
```

This message occurs when you attempt to "`logout`" or "`exit`" a shell and have one or more suspended jobs. The system doesn't want you to log out or exit the shell and kill the jobs unless you mean to. You can do a couple of things in response to this message. Use of the "`jobs`" command will tell you what job(s) you have suspended. You can choose to continue the job(s) in the foreground or background, and then log out, or, if you don't care if they terminate, you can just type "`logout`" again. Typing "`logout`" a second time with or without an intervening "`jobs`" command will result in logout and the termination of all suspended jobs.

A second possible message at logout is

```
% logout
Not login shell.
% _
```

As the shell can be invoked recursively, you can be "in" an instance of the shell that is not the one that was started when you first logged in. You must, however, be executing this login shell to log out. To terminate a shell execution, enter a CTRL-D sequence or type "`exit`". An additional "%" prompt character will appear on the terminal for every shell termination you effect by a CTRL-D or "`exit`". When you are in your original login shell, a CTRL-D will result in the message Use "`logout`" to logout. At this point, simply do what the message says! If you type "`exit`" when you are in your login shell, you will be logged out!

2.1.8 Password Maintenance

The "`passwd`" command can be used to change the password associated with your login name. To invoke it, type "`passwd`".

"`passwd`" will prompt you for your existing password, and then for the new one. The new password must be typed twice. Passwords must be at least six characters long. You should be very careful when choosing a password. It should never be something that would be easily guessed such as your name, your phone number, or the name of your best friend. Choose a "word" that is not in the dictionary. People have been known to write programs to guess passwords by trying every word in the on-line dictionary. Your best bet is to take two short words, such as "blue" and "cat" and put them together to make the password bluecat. This is relatively easy to remember, yet not vulnerable to a dictionary attack. To be extra secure, throw in a random upper-case character: bluEcat. This will be next to impossible to guess. You can also use numbers and non-alphanumeric characters such as * % $ # etc. Also:

- Never tell your password to anyone else.
- Never write your password down.
- If you suspect that someone knows your password, change it immediately.

2.1.9 On-line Manual

Volume 1 of the UNIX Programmer's Manual is kept on-line. This is a collection of files (one for each program) that tell you how to use all the different Unix programs. If you get stuck on something, you can look at some manual sections which might help. This is also useful for getting the most up-to-date information on a command. To print a manual section, type "man command-name". Thus to read up on the "comp" command, type:

```
% man comp
```

and, of course,

```
% man man
```

tells about the "man" command.

2.1.10 Files

A file can be thought of as something which contains data. This "data" can be the source of a program you're writing, the text of a document like this one, or any other collection of characters (such as directory information or executable code). A file always has a name associated with it which can be up to 255 characters long. Files are usually created with text editors.

A directory is a file containing information about other files and directories. A directory which is found "within" another directory is called a "subdirectory" and can contain even more subdirectories. A Unix file system is like a tree with a root (the "/" directory) and many branches (subdirectories).

When an account is created for you, you are assigned a "home directory". This is where you will find yourself every time you log in. If you were to type "pwd" (print working directory) after you logged in, you would see something like this:

```
% pwd
/users/admin/jsmith
```

which is the name of your "working directory" and the full "pathname" to your files. In this example, directory names are separated by "/"s. The pathname describes where your files are in relation to the total file system. When you create files, they will be "under" your home directory and their pathnames will look like this:

'/users/admin/jsmith/mklogin.c'

'/users/admin/jsmith/junk'

The "ls" (for list) command lists the names (not contents) of all the files in the current directory. (more on directories later)

The names are sorted into alphabetical order automatically, but other variations are possible. For example, the command "ls -t" causes the files to be listed in the order in which they were last changed, most recent first.

The "-l" option (that's the letter l, not the digit 1) gives a long listing. "ls -l" will produce something like:

```
% ls -l
total 2
-rw-r--r--  1 jsmith       205 Jan 30 11:24 myfile
-rw-r--r--  1 jsmith     76217 Jan 31 11:23 temp
% _
```

The date and time are of the last change to the file. The number before the date indicates the number of characters. The owner of the files, that is the person who created them, is jsmith. The -rw-r--r-- tells who has permission to read and write the file; in this case, the owner can read and write the files, but everyone can read them.

Options can be combined: "`ls -lt`" gives the same thing as "`ls -l`", but sorted into time order (i.e., the most recently modified files are listed first). You can also name the files you're interested in, and "`ls`" will list the information about only them.

The use of optional arguments that begin with a minus sign, like "`-t`" and "`-lt`", is a common convention for Unix programs. In general, if a program accepts such optional arguments, they precede any filename arguments. It is also vital that you separate the various arguments with spaces: "`ls-l`" is not the same as "`ls -l`".

The "`ls`" command doesn't normally list files whose names begin with a dot (or period), such as '`.login`', '`.cshrc`', and '`.bash_profile`'. If you want these files listed, include the "`-a`" option, as in "`ls -a`".

So far we have used filenames without ever saying what's a legal name, so it's time for a couple rules. First, filenames are limited to 255 characters, which is enough to be descriptive. Second, although you can use almost any character in a filename, common sense says you should stick to ones that are visible, and that you should probably avoid characters that might be used with other meanings. We have already seen, for example, that in the "`ls`" command, "`ls -t`" means to list in time order. So if you had a file whose name was "`-t`", you would have a tough time listing it by name. Besides the minus sign, there are other characters which have special meaning. To avoid pitfalls, you would do well to use only letters, numbers and the period until you're familiar with the situation. Note that there is a difference between upper- and lower-case letters in filenames. In this document, all filenames will be in lower-case.

There are special files, usually in your home directory, whose names start with a period (`.`). These files are not listed when you use the "`ls`" command unless you use the "`-a`" option. Filenames which begin with a comma (`,`) are removed automatically every night. Files beginning and ending with a sharp sign (#) are created by the GNU Emacs editor as temporary files and are deleted after three days.

2.1.11 Subdirectories

It is a good idea to create your own subdirectories in which to organize your files. This will make managing your files much easier. Let's say you want to make a directory '`src`' in which to put all the files containing sources of programs you have written. You type

```
% mkdir src
```

to create the subdirectory named 'src', and move the files into it by typing

```
% mv *.c src
```

which will move all the files with a '`.c`' extension to the '*src*' area. Now if you type "`ls -l`" you should see something like this:

```
% ls -l
total 3
-rw-r--r--  1 jsmith     56732 Apr  1 14:51 junk
-rw-r--r--  1 jsmith      1684 Apr  2 14:50 temp
drwxr-xr-x  2 jsmith        24 Apr  2 14:50 src
```

If you wish to move to your source directory, you can use the "`cd`" (change directory) command:

```
% cd src
```

Now your current directory is '`/users/admin/jsmith/src`'. Any files you create will be in this directory.

You can move to the directory immediately above you by typing:

```
% cd ..
```

If you use the "cd" command without any argument, you will be put back into your home directory. Another time saver in naming files is the use of the tilde (~). You can use this to prefix a user's home directory instead of typing out his full pathname. For example, to get a listing of the files under Fred Rated's directory, you can type "`ls ~frated`" rather than "`ls /users/admin/frated`".

A tilde without a username refers to your own home directory, so you can type "`ls ~`" to get a list of the files in your own home directory no matter what directory you may be in.

To remove the directory '`src`' (assuming that your current working directory is /users/admin/jsmith) type:

```
% rm src/*
% rmdir src
```

The first command removes all files from the directory; the second removes the now-empty directory. (You can't delete a directory unless it's empty.)

2.2 Using the Shell

The "shell" is the command interpreter: the program that listens to your terminal and understands the words you type as commands and arguments. The shell we use is called the C shell (csh)[1] because its command syntax is similar to the C programming language. Not only does the shell execute your commands, but it also has nice features such as filename completion, command aliasing, history substitution, and job control which will make using the system easier.

Using the shell is easy. For instance, if you type "`date`" followed by a Return, the shell will print out the current date and time on your terminal screen. The "`ls`" command will print out a list of your files on your terminal screen. Both of these commands send their output to the standard output device (or "stdout") which is usually the terminal. The standard input device ("stdin") is the terminal's keyboard.

2.2.1 Start-up

When you first log in, the shell looks for the files '`.login`' and '`.cshrc`' (for csh) or `.bash_profile` (for bash) and executes the commands in those files. These files make it possible for you to tailor your working environment to your specifications and to ensure that environment is always the same.

Why two files? The commands in '`.login`' are executed only when you log into the system. It sets up your terminal type and performs other one-time tasks. The commands in '`.cshrc`' are executed every time a new shell (csh) is started up, like when you fork out of a program back to the shell. The commands in '`.bash_profile`' are executed on startup of bash.

When you log out (by typing "`logout`"), the shell will run the commands in your '`.logout`' file.

2.2.2 Filename Completion

When you have the command "`set filec`" in your '`.cshrc`' file, the shell can complete the partially typed name of a file or user name. When you type the first few unique characters of a file name and then press Esc, the system will complete the filename for you. If it is not clear which file you want (the first few characters you typed are not unique to one filename), the terminal will beep at you. In bash, you can do the same without `set filec`.

In csh, if the partial filename is followed by CTRL-D, the shell will list all the matching filenames. It will then prompt you again, retyping the command line as you have typed it so far. In bash, you can type tab twice to achieve the same effect.

[1] actually, we use an enhanced version called tcsh, but for all practical purposes, we call it csh

2.2.3 History Substitution

History substitution allows you to use words from previous command lines in the command line you are currently typing. This makes it easier to correct simple errors in complicated command lines. To do this, the shell has a built-in mechanism that keeps track of some number of the commands you have typed. For example, the command "`set history=10`" in your '`.cshrc`' file will tell the shell to remember the last 10 commands you have typed:

```
% history
    1  6:33 ls
    2  6:33 date
    3  6:34 history
% _
```

This shows that I have given 3 commands: "`ls`", "`date`", and "`history`". The 6:33, 6:34 etc are the times those commands were typed.

A history substitution command begins with an exclamation point or "bang sign" (!). The command "`!!`" will repeat your previous command. If you give the "`date`" command and then "`!!`", the "`date`" program will run again:

```
% date
Mon Sep 16 14:21:01 PDT 2013
% !!
date
Mon Sep 16 14:21:04 PDT 2013
%
```

Here are some ways you can use history substitutions:

`!!`	re-run the previous command
`!`*n*	re-run command on line number *n* (from the history listing)
`!string`	re-run last command beginning with *string*
`^word1^word2`	In previous command, substitute *word2* for *word1*.

2.2.4 Aliases

The term "alias" means another name for something. Another feature of the C Shell allows you to give new, short names for long, frequently used commands with the "`alias`" command.

In csh, it is

```
alias newname 'long command string'
```

In bash, it is

```
alias newname='long command string'
```

For example, if you frequently use the "`find`" command to look through all your directories to find files named '`core`', you could make the following alias:

(csh)

```
% alias find_core 'find .  -name core -print'
```

(bash)

```
% alias find_core='find .  -name core -print'
```

Now you have a new shorter command ("find_core") that you can use to find 'core' files.

You can put the "alias" command in your '.cshrc' or .bashrc (or .bash_profile) file so that you can use them every time you log in. Aliases can be removed with the "unalias" command.

2.2.5 Using Files Instead of the Terminal

When you run a program, the program usually produces some output which appears on your terminal screen. If you wish, you can redirect the output into a file. For example, the "who" command will produce output like this:

```
% who
operator console Sep  3 05:59
eauu016  ttyp4   Sep  4 14:07 (csi-ts3.nts.uci.)
eaacct   ttype   Sep  3 15:25 (csi-ts3.nts.uci.)
nest     ttypf   Sep  4 08:18 (ghostbusters.nts)
eapu163  ttyq2   Sep  4 10:47 (csi-ts3.nts.uci.)
operator ttyq3   Sep  3 15:57 (csi-ts3.nts.uci.)
```

You can redirect the output from "who" into a file by typing:

```
% who > who_file
```

This command will create a new file called 'who_file' that contains the output from the "who" command. If you already had a file called 'who_file', its contents would be replaced by the output from "who". If you wish to append to an existing file, you can use two "greater than" signs:

```
% who >> who_file
```

2.2.6 Pipes

Pipes connect the standard output of one program to the standard input of another. To do this, just type the two commands with a "|" (vertical bar) between them.

Let's say that you want to see if your friend, John Smith, is logged in, but you don't want to look through all the output from "who". You can take the output from "who" and pipe it through "grep" like this:

```
% who | grep jsmith
jsmith ttyqo Jan 29 13:12
```

["grep" is one in a family of programs that search through a file (or other input) to find a matching pattern. Type "man grep" for more details.]

If he is not, you won't get any output.

Quite often it is useful to pipe the output from a different program to the "more" program. This will show you the output a page at a time.

2.2.7 Environment Variables

The shell lets you define a variable and assign a value to it. Many programs take advantage of this ability. For example, many programs that start up a text editor (such as "vi" or "emacs") look at the variable EDITOR to see which editor you wish to use. These variables are usually set in the '.login' or '.cshrc' (csh) or '.bashrc' or .bash_profile' (bash) files.

Here are some different ways of setting environment variables: (these are for csh, which is the default on the Sun instructional account)

```
setenv EDITOR emacs
setenv PAGER /usr/ucb/more
```

If you use bash (the default shell in Cygwin, recent versions of MacOS X, and Linux), then you need to adjust the syntax slightly:

```
export EDITOR=emacs
export PAGER=/usr/ucb/more
```

The shell also defines some variables for you such as `HOME`, which is your home directory, and `PATH`, which is the list of directories searched for executable files. The "`man`" pages for various programs will tell you which environment variables they use.

2.2.8 Filenames

The Unix filesystem looks like an upside down tree. There is one directory at the top called '`/`' (slash) or the "root" directory. All the other files in the filesystem are "below" this directory in many layers of subdirectories. If I type "`pwd`" (Print Working Directory) I will find out the name of my current directory:

```
% pwd
/users/physics/jsmith
% _
```

This shows that my current directory is called '`jsmith`' and it is "under" the directories '`/users`' and '`/physics`'. I can list the files in my current directory by typing "`ls`" and I can see what files are in a different directory by typing "`ls`" followed by a path. For example, what files are kept in '`/usr/local`'?

```
% ls /usr/local
IDEpipesh*      fanno@          nbbc*           rmf*
Pnews*          fc*             netcon*         rmm*
Rnmail*         finger*         newsetup*       rmt*
ali*            folder*         newsgroups*     rn*
anno*           folders*        next*           rprompt*
ansitar*        forw*           nmsh*           rrn@
apropos@        fstype*         nn*             rtar*
.
.
% _
```

[The directory name '`/usr/local`' is an "absolute" or "rooted" pathname because it starts with '`/`' (the root directory).]

There are certain shorthand ways of referring to files or groups of files without typing the full filename(s). The special characters that allow this are sometimes called "wildcard" characters. The most popular one is "*" which will match any or all characters. For example, "`ls f*`" will list all files in the current directory that begin with f, or "`ls a*b`" will list all files that begin with a and end with b.

`*`	matches any characters
`?`	matches any one character
`[abc]`	matches characters `a` or `b` or `c`

2.2.9 Job Control

Unix allows you to suspend a program you are running and then go back to it later on. This is called "job control". When you are using a program interactively, it is said to be in the "foreground"; once it is stopped (with CTRL-Z) it is in the "background". At this point you can put the job back into the foreground by typing "`fg`", or tell it to continue to run in the background with "`bg`". The shell gives a number to each job that is running in the background or that has been stopped. To see the list of current jobs, type "`jobs`". Each job will be listed, preceded by its number. The job most recently stopped is referred to as the current job and will have a + sign in front of it. You can kill a job by typing:

```
% kill %n
```

where *n* is the number of the job. A job left running in the background will stop when it needs input from the terminal.

2.2.10 Useful Commands

`alias` *name command*	Assign *command* to alias *name*. If *command* is omitted, the alias name is shown along with its current definition.
`bg` [%n]	Run the current (or specified) job in the background.
`cc` *file.c*	C language compiler. Translates C input files '*file.c*' into a machine executable program ('`a.out`').
`cd` [*dir*]	Change your working directory to '*dir*'. If *dir* is omitted, change to your home directory.
`cp` *oldfile newfile*	Make a copy of '*oldfile*' and call it '*newfile*'.
`exit`	Quit the current shell.
`f77` *file.f*	Fortran language compiler. Translates Fortran input files '*file.f*' into a machine executable program ('`a.out`').
`fg` [%n]	Bring current job or specified job (%n) into the foreground.
`fgrep` *string filename*	Search for pattern *string* in file '*filename*'.
`history` [*n*]	Display the history list. If *n* is given, display only the *n* most recent items.
`jobs`	List the jobs under job control.
`kill` *n*	Terminate process *n*, where *n* is determined with the "`ps`" command. If the job is running in the background, you can use the "`jobs`" command to find out the job number. In this situation the command would take the form: "`kill %n`".
`logout`	Terminate your login shell. Runs the commands in the file '`.logout`'.
`lpr` *filename*	Sends file '*filename*' to the lineprinter on the third floor of the CS building.
`ls`	List the files in your current directory. "`ls -l`" will list the files in "long" format showing size, modification date, permissions, etc.
`man` *command-name*	Gives information about the specified *command-name*. You can get more information about all the commands discussed in this section with the "`man`" command. Typing: "`man -k` *keyword*" will give you a short, one-line summary of all the commands dealing with that topic. If you wish to test this, try "`man -k` *information*".
`mkdir` *dirname*	Creates a directory with name '*dirname*'. The "`rmdir`" command is for removing empty directories.
`more` *filename*	Displays the file '*filename*' on your terminal screen one screenful at a time. Press Return to see one more line and press Spacebar to see one more screen at a time.
`mv` *oldname newname*	Changes the name of file '*oldname*' to be '*newname*'. You can also move a file into another directory with: "`mv` *file newdir*". This moves file '*file*' into the directory '*newdir*'.
`printenv`	Displays all the current environment variables and their values.
`ps`	Shows information about processes which are running under your terminal. Use "`ps x`" to find out about all your processes. This command is useful if you need to find out the process ID of a process in order to "`kill`" it.
`pwd`	Prints your current directory.
`rm` *filename*	Deletes the file called '*filename*'. You might not be asked for confirmation, so be sure you want to remove the file before you type the command. To require confirmation, "`rm -i` *filenames*" will ask you to type "`y`" or "`n`" before deleting a file.
`setenv` [*VAR* [*word*]]	When used alone (i.e. with no arguments) "`setenv`" will display all your environment variables and their values. With the *VAR* argument, the variable *VAR* will be given an empty (null) value. With both *VAR* and *word*, *VAR* will be given value "*word*". [Environment variables are normally given upper case names.]
`who`	See who is using the system. Try the "`w`" command for more info.

2.3 Files

2.3.1 File Protection

Unix provides a mechanism for permitting access to files and directories by other users. For every file and directory in the file system, there are three classes of users who may have access:

User (u) The user is the owner of the file, usually the person who created it initially.

Group (g) All users are assigned one or more user groups. Therefore, there is also a group ownership associated with each file.

Other (o) All users other than the owner of the file or a member of the file's group.

All (a) All users (User, Group, and Other). Each of these options is abbreviated to its first letter.

Let's examine the output from "`ls -alg`". The "`-g`" option causes "`ls`" to display the group ownership of a file. The "`-a`" and "`-l`" options cause all files to be displayed in long format.

```
% ls -alg
total 10
drwxr-xr-x  3 jsmith  admin    512 Apr  5 16:29 .
drwxr-xr-x  4 jsmith  admin    512 Apr  5 08:48 ..
-rw-r--r--  1 jsmith  admin   2315 Apr  5 16:29 mklogin.c
-rw-r--r--  1 jsmith  admin      0 Apr  5 16:29 mt
-rw-r--r--  1 jsmith  admin   2053 Apr  5 16:14 test.c
% _
```

The information in the left-most column (in this example, `d` or `-`) tells whether the entry is a directory or a normal file. The next nine columns are divided into three groups of three characters, giving the permissions for the User, Group, and Others, respectively. The three permissions are:

Read (`r`) A user with read permission for a file can look at the contents of the file. If the file is a directory, then the user can find out what files are contained in that directory using the "`ls`" command.

Write (`w`) A user with write permission for a file can change its contents. Having write permission for a directory allows a user to create or delete files contained in that directory.

Execute (`x`) A user who has execute permission for a file can use that file as a Unix system command. Execute permission for a directory allows access to the files contained within the directory, and it allows one to move to that directory.

The next number in the output from "`ls`" is the link count for the file, with which you need not concern yourself. The user and group ownership of the file are given next, followed by the size of the file in characters. The last information given before the name of the file is the date and time at which the file was last modified.

2.3.2 Changing Permissions

To change the protection on a file, use the "`chmod`" command. The first argument to the "`chmod`" command is the protection mode. This is followed by the names of the files to be protected. The protection mode consists of three parts: which class of user to work with (`u`, `g`, `o`, or `a`), whether to grant (`+`) deny (`-`), or set (`=`) access, and which protection to use. Each of these parts is a single character. For example:

`chmod g+r *.tex`	to grant read access to your group for all files in your account ending in "`.tex`,"
`chmod a-r` *resume*	To prevent anyone (including yourself) from reading the file '*resume*',
`chmod u+r` *resume*	Since you own the file, however, you can give read access back to yourself

2.3.3 Viewing Files

So how do you look at a file? There are a host of programs that do that. One is "`cat`" which copies the contents of files to the terminal: "`cat junk`" types one file, and "`cat junk temp`" types out two. In the second example, the files are concatenated and displayed onto the terminal. If they are long files, they will probably scroll right off the screen.

The program "`more`" allows you to look at a file one screenful at a time. (To use "`more`", type "`more` filename".) Use the Spacebar to go to the next screen when you see:

```
More--(n%)
```

Pressing the Return key will display the next line. Press h to get a list of the other commands available in "`more`". After you've become familiar with "`more`", you may wish to look at the "`less`" program which does even more than "`more`".

Chapter 3

The vi text editor

See also

- An Introduction to Display Editing with Vi, http://docs.freebsd.org/44doc/usd/12.vi/paper.pdf
- Vim Documentation, http://www.vim.org/docs.php
- 大家來學VIM(一個歷久彌新的編輯器), http://www.study-area.org/tips/vim/index.html

3.1 Introduction

The vi (visual) editor is a full-screen editor. This means that you will be able to position the cursor to the location in the file where you want to make your changes and additions. The changes that you make will be reflected on the screen. "vi" will work with many different types of terminals if you have set the terminal type correctly before invoking the editor.

Unlike "emacs" or "pico", "vi" has two modes: insert mode (in which you can enter text into a file) and command mode (in which you give commands to "vi"). "vi" also lacks the windowing and full-undo capabilities of "emacs". "vi" is, however, available on practically every Unix system.

More recently, "vim" (which stands for vi-improved) adds multiple undo, syntax highlighting, screening, and many more features. The path for vim is `/usr/local/bin/vim`, in case it is not in your path. In the following text, substitute vim for vi if you want to use vim.

3.2 Entering and exiting the vi editor

You invoke the vi editor by typing "vi filename". If the file exists, "vi" will display the first few lines of the file and give some status information about the file on the status line (the last line on the screen). If the file does not exist you will see something like this:

```
~
~
~
~
~
~
"letter.tex" [New file]
```

"vi" uses the the bottom line (the 24th line of most terminals) as its status line. It will display error messages, information on search strings, and various other informational messages on this line.

At the beginning of an editing session, "vi" makes a copy of your file and places it into a work buffer. All the changes and additions you enter will be added to the work buffer and not to the original file. If you are satisfied with the changes you have made, you can exit using the following command from command mode. ZZ

Command mode is entered by pressing the Esc key. If you are already in command mode, your terminal will beep at you. Don't worry, no harm will be done if this happens.

If you have made some drastic errors, you can exit from the editor without saving any of the changes. The command to do this is

```
:q!
```

which is entered at command mode.

3.3 Modes of Operation

"vi" has two modes of operation, insert mode and command mode. When you first enter "vi" you will automatically be placed in command mode. All of the "vi" editing commands are entered while in this mode. You can make certain you are in command mode by pressing the Esc key on your terminal. If you are already in command mode, no harm will be done by entering this mode again. You can exit command mode and enter insert mode by entering one of the insert commands.

3.3.1 Insert Mode

Insert mode allows you to insert text into the file you are editing. There are several ways to enter insert mode depending on what you want to do. They are:

- a append text after current cursor position
- i insert text before the current cursor position
- A append text after end of current line
- I insert text at beginning of current line
- o open a line below the current line
- O open a line above the current line

It is usually easiest to use i to insert text. If, however, you want to add text to the end of a line, you have to use A.

Once you have entered insert mode with any of the above commands you are able to continue typing and inserting text. "vi" will allow you to delete characters and edit within the current line you are editing but will not give you any other cursor positioning capabilities or editing functions beyond the ability to backup and delete characters and entered. In fact, if you try to use any of the cursor positioning keys, besides the DEL key or CTRL-W (used to backup over and delete words), while still in insert mode, you run the risk of losing portions of your text. Be certain that you are in command mode, not insert mode, when using the cursor positioning keys (please see the next section for a list of what these keys are).

If you need to do major editing on text, you can return to command mode by pressing the Esc key. You may now use all the cursor positioning commands and editing commands available in command mode.

3.3.2 Command Mode

Command mode is the mode of "vi" that you will use to perform most of your editing tasks. It is in this mode that you are able to position the cursor, delete text, change text, search for specific strings, substitute one string for another, exit from "vi", and perform a variety of other tasks.

Cursor Positioning

While in command mode you can position the cursor over any character on the screen. You can also display a different portion of the work buffer on the screen. You can move the cursor through the text by any Unit of Measure (i.e., character, word, line, sentence, paragraph, or screen). To move the cursor, just type the letter of the unit of measure. For example, to move ahead one word, type w.

l, SPACE BAR, →	one character to the right
h, ←	one character to the left
w, b	one word forward or backward (Punctuation counts as a word.)
W, B	one "blank-delimited" word forward or backward
j, ↓	one line down
k, ↑	one line up
)	beginning of next sentence
(	beginning of sentence
}	beginning of next section
{	beginning of section

Deletion

The x command deletes the character under the cursor. You can precede the x command by a repeat factor, i.e., a number that indicates the number of characters you wish to delete.

If you wish to delete words or lines, use the d operator. The d operator is used along with a repeat factor and a Unit of Measure. Here is a list of some useful delete commands and their results:

`d0`	delete to beginning of line (That's a zero!)
`dw`	delete to end of word
`d3w`	delete to end of third word
`db`	delete to beginning of word
`dd`	delete current line
`5dd`	delete 5 lines starting with the current line
`d)`	delete to end of sentence

On some terminals, deleted lines are replaced by an @ symbol at the left of the screen. These lines will not be written to the Work Buffer. If you wish to redraw the screen to get rid of the @s type CTRL-R while in command mode.

Changing Text

The c operator replaces existing text with new text that you enter. The new text does not have to occupy the same amount of space as the existing text. You may change one word to several words or a paragraph to a single character. The change operator c places a $ at the end of the text specified by the Unit of Measure and places "vi" in Insert Mode. Finish the change by pressing Esc. Below is a list of some useful change commands with their Units of Measure:

`cw`	change to end of word
`c3w`	change to end of third word
`cb`	change to beginning of word
`cc`	change the current line
`5cc`	change 5 lines starting with the current line

The Undo Command

The Undo command u undoes what you just did. It restores text that you just deleted or changed by mistake. "vi" will only restore the most recently changed text. If you deleted one line and then another with separate commands, the undo command will only restore the last line deleted.

The U command will restore the current line to the way it was before you started changing it, even if you have made several changes.

"vim" will let you undo as many times it can remember. Also, CTRL-R is the "redo" command in "vim." It lets you reverse an undo.

The Status Command

The status command CTRL-G displays the name of the file being edited, the current line number, the total number of lines in the Work Buffer, and the percent of the Work Buffer preceding the current line.

The . Command

The . (period) command repeats the most recent command that made a change.

3.4 Searching for a String

The forward slash / is used to find the next occurrence of a string, while you are in command mode. The syntax of the search command is:

`/string`

where you substitute the actual string you wish to search for instead of the word "string" above.

If you wish to find the previous occurrence of a string use the question mark instead of the forward slash. Notice that "?" and "/" usually appear on the same key on the keyboard. You can repeat the previous search without re-entering the command by using the n and N keys. The n key will repeat the last search exactly as entered and the N key will repeat the last search in the opposite direction.

3.4.1 Special Characters in Search Strings

Several characters take on special meaning when used in search strings. They allow you to search for strings in specific positions of the file, or allow you to perform wildcard searches (i.e., look for a match on any character within a pattern in your search string). These special characters and their functions are listed in the table at the bottom of this section.

Here are some examples: When the circumflex is the first character in the search string "vi" will look for the next line that begins with the string which follows. For example,

```
/^today
```

would find the next line that begins with today.

The dollar sign used after a string will find the next line that ends with that string. For example,

```
/?$
```

would find the next line that ends with a question mark.

The period . is used to match any character, anywhere in a search string. For example,

```
/t..e
```

would find the words time, tile, and mattress. It would find any word that contained a “t” followed by any two characters and then an “e”.

The characters “\>” are used to search for a word that ends with a specific string of characters. For example,

```
/ing\>
```

would find the next word that ends with “ing”.

The characters “\<” will search for a word that begins with with the string entered. For example,

```
/\<9Lives
```

will find the next word that begins with “9Lives”.

Square brackets surrounding two or more characters will match any single character located between the brackets. For example,

```
/dis[ck]
```

would find the next occurrence of either “disc” or “disk”.

3.4.2 Search String Special Characters

Symbol	Meaning
^	matches beginning of line
$	matches end of line
.	matches any single character
\>	matches the end of a word
\<	matches the beginning of a word
[ab]	matches any of the characters between the brackets (a or b)

3.5 Issuing ex Commands

“vi” is actually a screen-oriented extension of the “ex” editor. It allows you to use “ex” commands by preceding them with a colon :. When you type :, “vi” will print a colon at the beginning of the last line of the screen and allow you to type the “ex” command. If you change your mind and don’t want to issue an “ex” command, just press the DEL key until the cursor is back in the text.

We’ll be discussing three of the many “ex” commands available: the Substitute, Read, and Write commands. Substitute is described in the next section; Read and Write are described in another section. The range of “ex” commands is too large for this guide. For more information, consult the “Ex Reference Manual” and the “Edit/Ex Command Summary”.

3.5.1 String Substitution

The Substitute command is a combination of a search command and a change command. The operator for the Substitute command is [s]. It searches for a string, and when it finds it, it replaces it with the string you specify. The syntax of the Substitute command is:

`:[address]s/search-string/replace-string[/g]`

If you do not specify an address, the Substitute command will only search the current line. The “g” indicates that you wish to make this substitution globally; i.e., each occurrence of the search-string on a line will be replaced by the replace-string (as opposed to just the first occurrence).

The Substitute Address

The following characters are used to compose an address:

n	Line number *n* will be searched
n,*m*	specifies a range of line numbers (*n* through *m*) to be used in the search. The range is inclusive.
.	represents the current line
$	represents the last line of the work buffer
1,$	represents the entire work buffer
%	another way of saying 1,$

Examples of String Substitution

```
:s/larger/largest
```

replaces the string "larger" on the current line with the string "largest".

```
:1,.s/Section/Chapter/g
```

replaces every occurrence of the string "Section" with the string "Chapter" from line 1 in the file through the current line. The "/g" indicates that "Section" should be replaced by "Chapter" every time on a line.

3.6 The Use of Buffers

One useful function of an editor is to move and copy text from one position in a file to another, or to insert a different file into the file you are currently editing. "vi" gives you this capability through the use of buffers. "vi" has a General Purpose Buffer and 26 Named Buffers that you can use to hold text during an editing session. Whenever you use a delete command, the text you delete is automatically placed in the General Purpose Buffer. You can put this text at another location in your file by the use of the Put commands (p and P).

The contents of the General Purpose Buffer are reinitialized every time you issue a change, delete, or yank command (yank will be described below). You can put the text from the General Purpose Buffer into several different places in the Work Buffer as long as you do not issue one of these commands between Put commands. The use of named buffers comes in handy when you would like to move text around in your file but do not want to worry about accidentally writing over text in the General Purpose Buffer. The 26 named buffers are named by the 26 letters of the alphabet (lower-case letters only). You can store 26 different blocks of text which can be used by later put commands. To specify which named buffer to use, type " and the letter of the buffer before typing the change, delete, or yank command.

3.6.1 The Yank Command

The Yank command y is identical to the delete command except it does not delete text from the Work Buffer. It simply makes a copy of the text in the Work Buffer and places it in the General Purpose Buffer or one of the named buffers. The syntax of the yank command is the same as that for the delete command except you substitute a y for the d in all commands. Here are some examples:

yy	make a copy of the current line and place it in the General Purpose Buffer
"gyy	make a copy of the current line and place it in the named buffer *g*
"a5yy	copy the next 5 lines and place them in the named buffer *a*

3.6.2 The Put Command

After you have deleted, changed, or yanked text from the Work Buffer you can place it elsewhere in your file by using the Put command. The Put command uses the p and P operators. The p operator places characters and words from the buffer after the current character, and places lines of text in the buffer after the current line. The P operator places characters and words before the current character, and lines of text before the current line. Here are some examples:

`p`	Will put the contents of the General Purpose buffer after the current character or line (depending upon whether the buffer contains a character, word or line).
`"gp`	Will put the contents of the named buffer g after the cursor
`"aP`	Will put the contents of the named buffer a before the cursor

3.7 Reading and Writing Files

"vi" allows you to incorporate the contents of another file into your current Work Buffer. You are also able to create a new disk file consisting of all or part of your current Work Buffer. To do these two operations use the Read and Write commands.

3.7.1 The Read Command

The Read command :r will read the contents of a disk file and insert it into the Work Buffer following the current line. The syntax for the command is

`:r` *filename*

where *filename* is the pathname of the file that you want to read.

3.7.2 The Write Command

The Write command :w will write all or part of the Work Buffer to a file. The syntax for the command is

`:`[*address*]`w[!]` [*filename*]

If you do not use an address or a filename, the entire work buffer is written to the file you are editing, hence it gives you a way of permanently saving the changes you have made to your file without actually exiting from "vi" with the ZZ command. The address is specified in the same way as in the Substitute command and specifies the portion of the Work Buffer to be written. If no address is included, the entire Work Buffer is written. The exclamation point is necessary unless you are writing out the entire contents of the Work Buffer to the current file being edited. It forces "vi" to perform the command. You are able to append to an existing file by using the following form of the command:

`:`[*address*]`w>>` *filename*

3.8 Word Abbreviations

Vi lets you define abbreviations that can be expanded into another phrase. For example,

```
:ab eecs Electrical Engineering and Computer Sciences
```

Whenever you type "eecs" in insert mode as its own word (not part of another word), it is automatically expanded into "Electrical Engineering and Computer Sciences". To undefine this abbreviation, just do

```
:unabbreviate eecs
```

3.9 VIM-Specific Features

Vim (vi-improved) has a many features that are missing in vi. The best way to find out is to go through the vim tutorial, which is part of vim. You can type

```
:help
```

from inside vim.

Here we will just show a number of useful features.

3.9.1 Syntax Highlighting

Vim "understands" a number of languages and can do syntax highlighting. What this means is that it can use different colors to display different kinds of elements in the language. It is usually helpful to see keywords, identifiers, strings, comments, and operators in different colors. Vim will know what language it is, based on the file extension. For C files, the extension is ".c"; for python files, the extension is ".py".

To turn on syntax-highlighting, do

```
:syntax on
```

(If you want syntax highlighting to be always on, just create a file named `.vimrc` in your home directory, and put `syntax on` in there without the colon.)

3.9.2 Unix, Mac, or DOS File Format

Text files may use Unix, Mac, or DOS file formats. The difference is the return character, whether it uses CR (carriage return), LF (line feed), or CRLF (carriage return followed by line feed). DOS uses CRLF, which is the most verbose. VIM and many modern text editors will handle all of them correctly, but some other might not. VIM will show you what type of file format it is using. What you can do is

```
:set fileformat
```

And it displays what file format it is. It actually does not set the fileformat, but merely displays it.

To force the file format to the one you want, you just type the command with = and the format name. For example,

```
:set fileformat=unix
```

This forces the format to be unix, if not already. The options are unix, dos, mac.

3.9.3 Unicode and other encodings

Unlike vi, which is primarily for ASCII terminals, Vim is aware of different encodings and character sets, both roman and nonroman. Some encodings such as ASCII and other roman character sets use a single byte per character; however, east Asian characters tend to use two bytes per character.

More recently, unicode which is a way to bring together all the character sets into one encoding, uses two bytes per character internally. However, to maintain compatibility with existing ASCII-based programs, unicode itself is almost always *embedded* in another coding scheme. The most common embedding is called `utf-8`. This is a variable-length encoding. For the ASCII subset, utf-8 looks identical to ASCII in that it uses a single byte per character. To handle other characters, it uses two bytes for some characters, and three bytes for other characters. The first byte will determine how many more bytes to follow.

To tell vim to set the encoding (to utf-8, say), do

```
:set encoding=utf-8
```

Chapter 4

Setting up SSH Public/Private Keys

SSH (Secure Shell) can be set up with public/private key pairs, so that you don't have to type the password each time. Because SSH is the transport for other services such as SCP (secure copy), SFTP (secure file transfer), X Windows, and other services (CVS, etc), this can be very convenient and save you a lot of typing.

A more comprehensive how-to for setting up instructions for Windows and Linux can be found at http://inside.mines.edu/~gmurray/HowTo/sshNotes.html.

For a detailed explanation on SSH-related topics, see 遠端連線伺服器 SSH / XDMCP / VNC / RDP http://linux.vbird.org/linux_server/0310telnetssh.php

4.1 SSH Version 2

Most of you have SSH version 2. On the local machine, type the BOLD part. The non-bold part is what you might see as output or prompt.

- Step 1:

```
% ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key (~/.ssh/id_dsa):  (just type return)
Enter passphrase (empty for no passphrase):  (just type return)
Enter same passphrase again:  (just type return)
Your identification has been saved in ~/.ssh/id_dsa
Your public key has been saved in ~/.ssh/id_dsa.pub
The key fingerprint is:
Some really long string
%
```

- Step 2:
  Then, paste the content of the local `~/.ssh/id_dsa.pub` file
  into the file `~/.ssh/authorized_keys` on the remote host.
  That's it!

FAQ

- Q: I follow the exact steps, but ssh still ask me for my password!
- A: Check your remote .ssh directory. It should have only your own read/write/access permission (octal 700)

```
% chmod 700 ~/.ssh
```

4.2 SSH Version 1

- Step 1:

```
% cd ~/.ssh
% ssh-keygen -t rsa1
Generating public/private rsa1 key pair.
Enter file in which to save the key (~/.ssh/identity):  (just type return)
Enter passphrase (empty for no passphrase):  (just type return)
Enter same passphrase again:  (just type return)
Your identification has been saved in ~/.ssh/identity
Your public key has been saved in ~/.ssh/identity.pub
The key fingerprint is:
Some really long string
%
```

- Step 2:
  Then, paste content of the local `~/.ssh/identity.pub` file
  into the file `~/.ssh/authorized_keys` on the remote host.

Chapter 5

Typescript

A typescript is a text file that captures an interactive session with the unix shell. Very often you are required to turn in a typescript to show that your program runs correctly. To create a typescript, use the `script` command. Here is an example:

- Type the command

  ```
  script
  ```

  into the shell. It should say

  ```
  Script started, file is typescript
  % _
  ```

  This means it is recording every key stroke and every output character into a file named "typescript", until you hit `^D` or type `exit`.

- Type some shell commands; you can also enter the python prompt. But don't start a text editor!

- Stop recording the typescript by typing `exit`.

  ```
  % exit
  Script done, file is typescript
  % _
  ```

- Now you should have a text file named `typescript`. Make sure it looks correct.

  ```
  % more typescript
  Script started on Thu Sep 25 23:43:56 2003
  ...
  ...
  ```

You should immediately rename the typescript to another file name. Otherwise, if you run `script` again, it will overwrite the `typescript` file.

Note: If you backspace while in script, it will show the `^H` (control-H) character in your typescript. This is normal. If you use `more` to view the typescript, then it should look normal.

Chapter 6

Cygwin Installation and Setup (for Windows)

Cygwin is a Unix-like environment for Windows. It allows you to access many command-line tools and some graphical user-interface programs in very similar if not the same way as command-line shells on other Unix systems such as the workstation clusters, Linux, and Mac OS X.

This chapter is for Windows users only. If you use Linux or Mac OS X, you should just use the terminal program that comes with the standard installations.

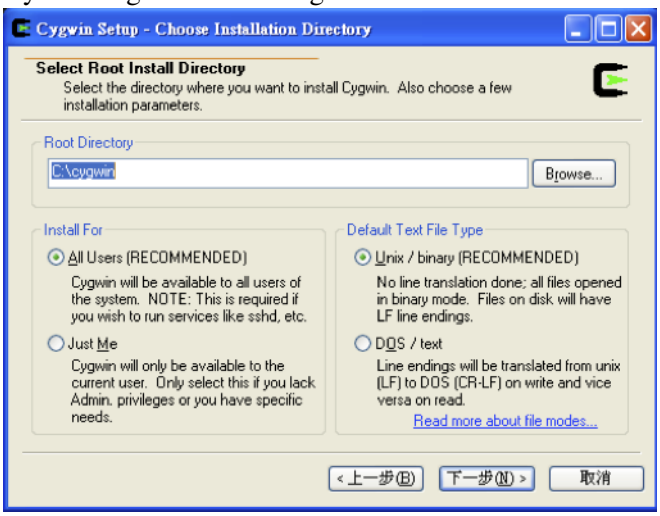
Having Cygwin is very useful, because it allows you to run many unix commands locally on your own machines when you are not connected to the Internet. You can also install many packages, ranging from C compiler, ssh client, Vim text editor, and just about anything in this environment. Note that PuTTY is only an SSH client program that lets you connect to a remote machine, but it cannot let you work locally. Cygwin, on the other hand, lets you work locally, and it also can run an SSH client program that lets you connect to a remote server, just like PuTT does.

Step 1

Open your web browser to `http://cygwin.com/`. Download the setup program, which may be `setup-x86.exe` or `setup-x86_64.exe`, depending on your system. (Previously, it was just setup.exe)

Step 2

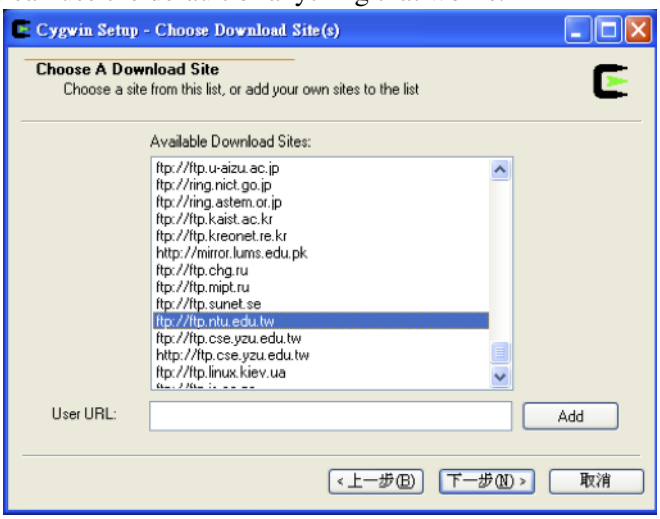
Double click on setup.exe and you will get the following window



Choose your installation directory and click Next step.

Step 3

Choose a download site. You can use the default or anything that works.



Some sites are faster than others. Try to choose something that is hosted at a domestic university.

Step 4

Press “View by Category” and install packages.

Minimally, it is recommended that you install

- gcc (Gnu C Compiler). Specifically,
  - Develop > clang: C/C++/ObjC Compiler frontend based on LLVM
  - Develop > gcc: GNU Compiler Collection
  - Develop > gcc-core: GNU Compiler Collection (C, OpenMP)
  - Develop > gdb: The GNU Debugger
  - Develop > mingw-gcc-core: GNU Compiler Collection (C, OpenMP)
- vim (vi-improved text editor). it is at
  - Editors > vim: Vi IMproved - enhanced vi editor
- ssh (secure shell). it is at
  - Net > openssh: The OpenSSH server and client

Click on the

You can install other packages later by running setup.exe again.

Step 5

Actual installation. This make take a while, because it has to download everything from the Internet. You may want to bring your computer to the campus and get faster download speed.

Step 6

When finished, you will get a shortcut that takes you to the command (bash). You can follow the unix guide in Chapter 2 with some minor differences. First, the prompt in bash is the $ character rather than % in csh. Second, instead of `setenv` in your .cshrc file, you use `export` in your .bash_profile instead.

Chapter 7

Xcode and GCC Installation (for Mac OS X)

If you want to run GCC on a Mac OS X computer (running Mountain Lion, or Mac OS X 10.8 as of this writing), you may have to install it separately. The easiest way is to install Xcode first, and then use Xcode to install command-line tools (which includes GCC and related files).

7.1 GCC Installation through Xcode

Xcode is Apple's integrated development environment for programming Mac OS X, iOS, and just about all the languages that are supported in the open-source community. Using Xcode is probably the easiest way to install GCC.

- First, launch "App Store" and search for "Xcode" in the upper-right-hand search box.
- Second, click on the "Free" button next to the Xcode icon to install it.
- Third, launch Xcode, and go to `Xcode menu > Preferences > Downloads`.

Fig. 7.1 shows a screen shot of the Command-Line Tools installation option in Xcode.

7.2 GCC Installation without Xcode

Alternatively, there is a separate installation package available via https://developer.apple.com/downloads/index.action?=Command%20Line%20Tools%20%28OS%20X%20Mountain%20Lion%29.

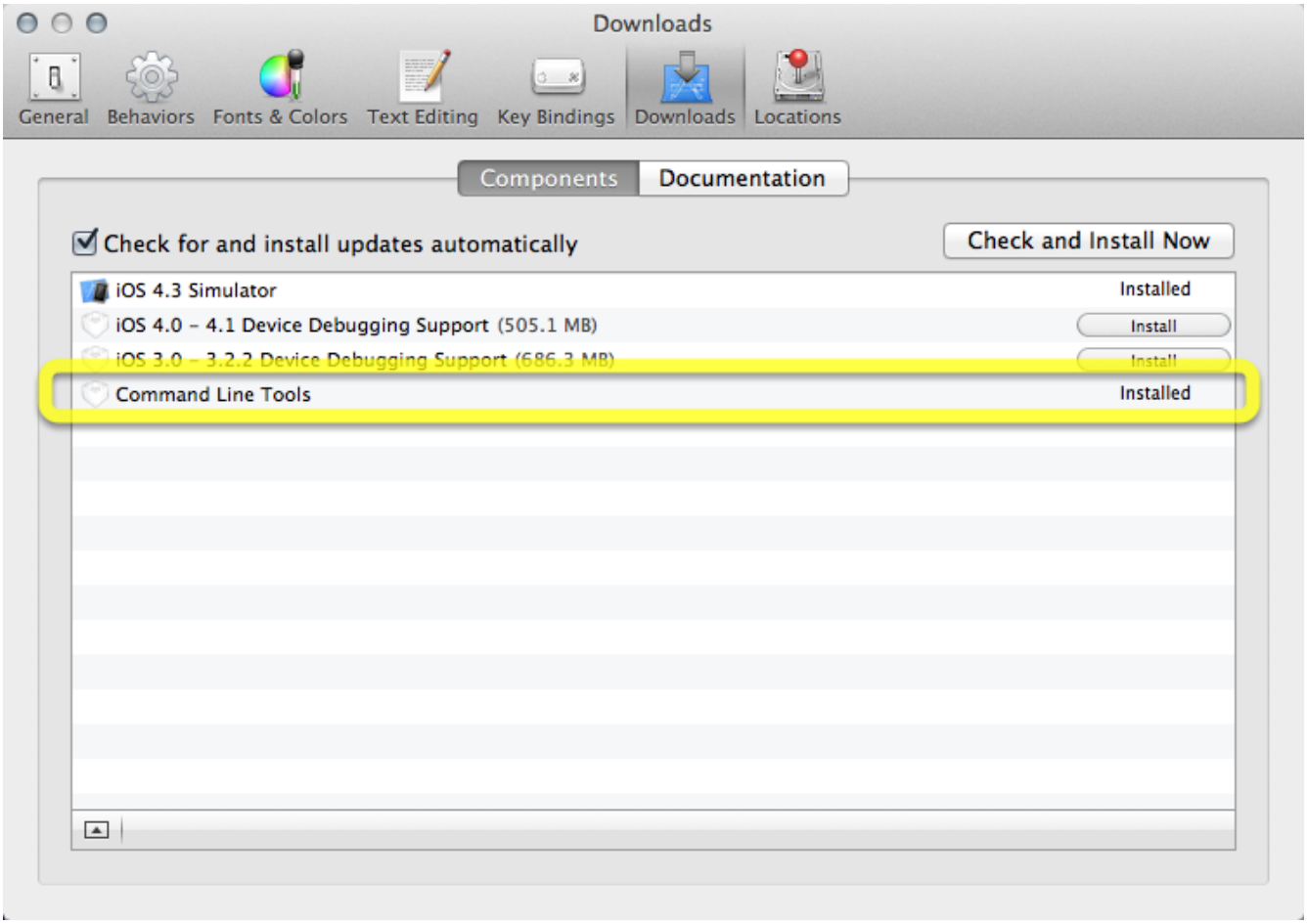


Figure 7.1: Xcode dialog to install command-line tools, including GCC.