UNIX

Every computer you use has an Operating System *which completes the tasks you send it with menu selections, mouse clicks, and typed commands. The operating system of the NeXT computer is, as you might have guessed, UNIX.*

Up until now, you may have only used menus, windows, and buttons in your interaction with UNIX. This Graphical User Interface (GUI) is one of the NeXT's strengths. But a far more common interface that you will find available on every UNIX system is the Command Line Interface (CLI). In this situation you exchange information with the computer solely through text input and output — you type commands and the computer prints responses.

Why would you want to work with just text when you have fantastic menu-driven programs at hand? That's for you to find out in this course. Suffice it to say that many people enjoy the power and flexibility of UNIX's Command Line Interface, and prefer to use it over menus, windows, and mice wherever possible.

Hopefully the following information will get be enough to get you started. Don't worry about learning every single thing... just absorb as much as you can and use this CheatSheet for reference as you work with UNIX in the labs.

Where to get help

Manual Pages

These handy guys may well be one of your most useful assets for finding UNIX information quickly. In a Terminal window, simply type:

```
man <command>
```

This will bring up help on the command you type in place of <command>.

If you want help on a particular subject, use the following:

```
man -k <subject>
```

This will bring up a list of commands relevant to <subject>. You can then use the normal `man` command above to get more information on any of those listed.

CIT documentation

The packet from CIT entitled, "Introduction to Phoenix" has many nitty-gritty details that all of us often forget. It is a good thing to bring to lab for reference. Anyway, we all know how much fun these things are to read when you get bored. You can pick these up in the CIT InfoCenter, at 87 Prospect Avenue.

Concepts

Userids and addresses

Every UNIX user has a unique userid. From the computer's point of view, your userid is your only real name. People who want to contact you from outside Princeton must type `<your userid>@phoenix.Princeton.edu` in order to reach you. This whole "phrase" is your own unique *Internet address*.

Passwords

Nothing but common sense here. Everyone has their own password, and *no one* should know yours. When you choose a password, *don't use a real word.* If you do, there's a chance that someone will discover it, because with a little UNIX knowledge, it's not too hard to write a program that will discover every password that can be found in a dictionary.

Why does it matter if someone guesses your password? Unfortunately, there are scum who would love to enter your account, using your password, and break into other computers. You would be blamed.

If you want to change your password right now, ask your TA for help changing it with the Preferences application.

Files

Files are "collections of related data records" (Webster's). They are more of an intuitive concept than anything else. A word-processing document, for example, is a file.

Directories

UNIX allows you to organize your files into directories. A directory is like a folder, which (usually) contains related files. You can have directories within directories, forming an entire "tree" of directories. (If you were to draw out the directory structure on paper it would resemble a genealogical tree.)

Along with every other UNIX user, you have a **home directory**. This is where you keep all your own files (and directories). It's also the place you find yourself when you log in. The computer's shorthand for "your home directory" is "~", and its shorthand for someone else's home directory is `~<userid>` . For example, you would refer to `omkensey`'s home directory as `~omkensey` .

Whenever you want to run an executable file, you usually need to tell the computer exactly where the file is. For example, throughout this course you will use the `handin` program to submit your work electronically. Since `handin` is in the **~cs111/bin** directory, to run it you would type

```
~cs111/bin/handin.
```

Luckily, there's another way. If you type a command which isn't in the current directory, the computer will search through a list of other directories that contain commonly-used commands. This list is called your *path*. If **~cs111/bin** is in your path, you could merely type

```
handin
```

to run the program. To see your list of paths, type `echo $path` .

For more information on files and directories, see “Welcome to NeXT.”

Disk quotas

Your *disk quota* is the maximum amount of information you are allowed to store in your home directory. To get information about how much storage you have used, and how much space you have left before you hit your disk quota, type `quota -v` .

If you use the `quota -v` command, you will see a number labelled “limit.” This is the absolute, unyielding maximum amount of information you can store. If you approach this limit, the computer will let you know, and will ask you to remove some files.

Commands

This section will give an overview of some of the more useful UNIX commands. It is by no means comprehensive, but it should give you what you need to know for basic survival. To use these commands, type them while in a Terminal window (see the Terminal CheatSheet).

Necessary commands

- Type **`ls`** to get a listing of what is in your current directory. You should know that UNIX hides files beginning with dots (like **.login**) — this is just a convenient way to hide files you know are there and don’t want to see each time you type `ls`. If you *do* want to see the hidden files, type `ls -A`.

  While `ls` gives you just the listing, `ls -s` will tell you the size of each file:

  ```
  prompt> ls -s
  total 1001
      4 hummingbird.wn     1 now.wn         1 swan2.wn
    411 hurrah.snd         2 swan1.wn     584 were_free.snd
  prompt>
  ```

  The file sizes next to each file are given in kilobytes(KB) — roughly a thousand characters. A single-spaced page in WriteNow, like **hummingbird.wn** above, weighs in at about 4 kilobytes.

  One important point: if `ls -s` comes across a directory, it *will not* tell you the size of everything inside it. It will give you something like 1 KB — that’s how much space the directory needs to keep an index of everything that’s inside.

  Finally, to see which elements of a listing are directories and which are executable programs, type `ls -F` . Directories are followed by a slash (“/”), and executable files are followed by an asterisk (“*”). In the following example, **TA** is a directory, **mumble** is a normal file, and **hello** is an executable file.

```
TA/                        mumble                        hello*
```

- **`cd`** is used for changing directories. To move into a directory, just type `cd <directory name>`. Typing `cd ..` will take you up to the directory which contains the current one. If you get lost, `cd` by itself will take you back to your home directory.

  REMEMBER THIS: *use ~ as an abbreviation for your home directory. Typing*

  ```
  cd ~/songs
  ```

  will send you to the subdirectory `songs` *in your home directory. You can also get to someone else's home directory by typing* `cd ~<userid>`*, where <userid> is the userid of the person you are looking for.*

- **`cat`** spits out the contents of a file or files. Type `cat <filename(s)>` and watch the file(s) scroll up the screen, one after the other. (This is where the command got its name... from con*cat*enating files and sending them to the screen.)

- **`more`** lets you view a file one screenful at a time. Type `more <filename(s)>` to view the files in <filename(s)> in small chunks. Pressing the spacebar will take you to the next page.

- **`rm`** will erase any files you tell it to. Type `rm <filename(s)>`. Be careful with this command! If you do erase something by accident, all is not lost. Send mail with a request to restore your file to sunadmin@phoenix.

- **`cp`** copies a file to a new name or place. The format is

  ```
  cp <original file> <new file>.
  ```

  Typing `cp ~/wolves ~/lower` will make a copy of the file **wolves** in your home directory and will name the copy **lower**.

- UNIX has a fast way to send a file to the printer: **`lpr`**. If you just type

  ```
  lpr <filename>
  ```

  your file should go to the nearest printer (the "local" printer).

  If that doesn't work, or if you want to send your file to a specific printer, use

  ```
  lpr -P<name of printer> <filename>.
  ```

  For example, `lpr -Pnext_cs_001_toimpossible laughing` would send the file **laughing** to the printer to the left of the NeXT machine "toimpossible." See the Printing CheatSheet for more on printer names.

- To print a file with half the paper `lpr` uses, use the `enscript` command. The format is the same as that of `lpr`, except after `enscript` type -2rG (**2** columns, **r**otated 90 degrees, in **G**audy, fancy, format). For example,

  ```
  enscript -2rG -Pnext_cs_001_toimpossible laughing
  ```

Not-absolutely-necessary-but-still-very-interesting commands

- `finger` will tell you information about a user. Give it a try by typing `finger <name>`.
- Typing `who` will tell you who is currently logged onto the system and where they are logged in from.
- `w` is like `who`, but the last column of its output tells you what each person is doing. Try it — it's legal to be a voyeur on the network!

Advanced commands

- `grep` is not a command; it's a way of life. In its most simple usage, `grep` finds a search-string in a text file. Type `grep 'foo' <filename(s)>` to find the string "foo" in the file(s) <filename(s)>. `Grep` will output every line containing "foo" to the screen.

  To find "flowers" in the files **guatemala** and **amanita**, type

  ```
  grep 'flowers' guatemala amanita
  ```

- `wc` counts words, lines, and characters in a file. Type `wc <filename(s)>`, and `wc` will give you something like this:

  ```
  toyou% wc foo bar
      904    7950   51288  foo
      165    1348    8432  bar
      1069   9298   59720  total
  toyou%
  ```

  The first column is the number of lines, the second is the number of words, and the third, the number of characters.

- `compress` shrinks a file's size down to, on average, about a half of its previous size. There are many reasons for compressing files. One is simply to save disk space, which is expensive and sometimes scarce. The other is to make a file smaller so it can be transmitted faster over the network. Compressed files all have a `.Z` on the end. To compress a file, type `compress <filename>`.
- `uncompress` uncompresses a compressed file. Any file which ends in `.Z` is compressed. To uncompress it, type `uncompress <filename.Z>`.

For those who want to become UNIX wizards

Pipes

Herein lies the beauty and power of UNIX. With a pipe (|), you can send the output of one command into the input of another. For example, if you come across a directory containing a huge number of files, you might type

```
ls | more
```

This will send the output of the `ls` command into `more`, which lets you view it one screenful at a time.

If you think of each UNIX command as a flow of data, pipes are the means of controlling that flow. Normally, the effect of a command flows right out onto the screen, the simplest case of data flow. By using a pipe, you can make that flow of data go someplace else...to another UNIX command, for example. This is the area in which `grep` truly shines. Let's look at another example.

The `w` command lets us see what each person on the system is doing at the moment. Unfortunately, if you are on a large UNIX system such as phoenix, there are bound to be many, many users. It becomes tedious to look through all of the people to find your friends each time you check up on them. `Grep`, as we know, will find any lines containing a certain string. So we want to take the flow of data from `w` and "pipe" it into `grep`, like this:

```
w | grep <name of friend>
```

This series of commands will execute `w`, but instead of printing its output on the screen, gives it to `grep`. `Grep` in turn looks for your friend's name and since there is no pipe after grep, the flow goes onto the screen, printing out every line with your friend's name in it. Voilá!

Output redirection

- This isn't too hard. If you follow a command with a greater-than sign (>) and a filename, the output of the command is saved into that file. For example,

  ```
  ls > listing
  ```

  saves the output of the `ls` command into a file called **listing**.

Filename expansion

- This isn't too hard, either. Say you want to count the words of every file in your current directory. Luckily, you have a special symbol which makes it easy. The ***** symbol "expands" to mean any combination of characters. So typing

  ```
  wc *
  ```

  would count the words in all the files of the current directory.

- Here's another example: typing

  ```
  grep 'buck' a*
  ```

  will search for the string "buck" in all the files that begin with "a."

Aliases

- UNIX allows you to build "aliases" for those long commands which you find yourself typing over and over again. For example:

  ```
  alias fb "who | grep foobar"
  ```

  will check to see if foobar is on the system whenever you type `fb`.

- You can "pass arguments" to the alias with the sequence \!* . If you type

  ```
  alias hello "who | grep '\!*'"
  ```

the alias will substitute, in place of \!*, anything you type after the `hello` command. So if you type

```
hello eedecker
```

the computer will actually receive the command “who | grep ‘eedecker’”. (This command finds out who is on the system, and only prints out `eedecker`, if she is on.)

- That’s all! To make an alias permanent, type the alias command (as above) at the end of a file called **.cshrc** in your home directory. The **.cshrc** file is run every time you log in.

Regular Expressions

Say you want to search a text file for every line which begins with ‘a’. You can’t just use `grep` *to search for ‘a’ — you’ll get hundreds of lines! Fortunately, grep accepts certain special characters — like one that represents the beginning of a line — in its search-strings. Expressions which use these special characters are called **regular expressions**, and they pop up often in UNIX.*

Here is a tiny, incomplete list of regular expression characters and what they do. Include any of these in search strings just as you would include a normal character.

^	at the beginning stands for the beginning of a line. To find all lines in the file **foo** that begin with the letter “a”, you might type `grep '^a' foo`.
$	at the end stands for the end of a line (it “anchors” the expression to the end), and it’s used just like ^. Type `grep 'a$' foo` to find every line in **foo** which ends in “a”.
.	stands for *any* single character, so ’r.m’ could mean ’ram’ or ’rem’.
.*	will match *anything*. So m.*e fill find *anything* which starts in m and ends in e.

For a more complete list, look for regular expressions in the `man` *page for* `ed`.

UNIX Cultural note: In the old days, UNIX users had to use `ed` as a text editor (*you* have Edit or `vi`). `ed` is only cute until you try it. In `ed`, you can’t look at your document while you edit it. Instead, you type commands to edit your file, and use your imagination to see how it looks as you make changes.

Anyway, one of `ed`’s commands searches for text, and accepts regular expressions. The command is g/<regular expression>/p, and it does the same thing as grep. Actually — and this is where the culture shows itself— “grep” can be found in the abbreviation for ed’s search command: “g/re/p”.

Another related note... some UNIX wizards know `ed` so well that they use it by choice. To them, arrow keys and pointing devices are cumbersome and annoying.