

Shell Basics

Jason Wilcox

CS Tutors – Portland State University

May 3, 2005

This document will teach you the basics of using shells within the Unix environment. Each time you login to your account you will be presented with a shell, and this will be the jumping off point for almost all of your work.

1 The Prompt

Before you tell the shell what program, or command, to run it will present you with a prompt waiting for input. At this prompt you can type in the name of the programs you wish to run and it will begin their execution for you. This prompt will look similar to the following:

```
203 lesath.cs.pdx.edu>
```

After you type in the command you wish to run at this prompt you will need to press the return key to actually begin the command's execution.

2 Changing Shells

There is not a single shell in UNIX, but instead there are many available to choose from; some popular shells include `tcsh`, `bash`, and `zsh`. When your account is first created your shell is `csh`, but it is severely crippled so we will show you how to change it in this section. It is strongly recommended that you switch your shell to `tcsh` because of how our system is configured.

To switch your shell you simply execute the command `chsh`. You will then be prompted for your password, and after that present with a prompt asking you to specify your new shell. Type `/bin/tcsh`, followed by return, and on your next login the shell you use will be changed. The following excerpt shows how the entire process will appear at your terminal.

```
202 lesath.cs.pdx.edu> chsh
Password for jasonw: PASSWORD FOLLOWED BY ENTER
Changing shell for jasonw on walt.ee.pdx.edu
Old shell: /bin/csh
New shell (? for help): /bin/tcsh ENTER
```

3 Re-Executing Commands

After you have typed several commands you may wish to re-execute them, but typing them in could be tedious and time-consuming; for example, you may be running your editor and compiler many times during a typical coding session. Luckily, several shells provide ways to avoid this tedium.

The most basic way to re-execute commands is through the arrow keys on the keyboard. *Note: this will not work if `cs`h is still your shell.* The *up-arrow* will cycle through your commands from most recently executed to most recently executed. As an example, assume you had recently done a `mkdir`, followed by a `cd`, and finally an `ls`. The first time you press the *up-arrow* the `ls` command will reappear on the prompt, the second time it will be replaced by the `cd` command, and a third *up-arrow* will get you back to the original `mkdir`. You can also use the *down-arrow* to move forward through your commands, such as going back from the `mkdir` to the `ls`.

This method works well if you have recently executed a command so only two or three up arrows will get you there. However, it will not be all that convenient if you executed the command 20 or even 50 commands back. To do that the `!` notation and history command come in handy. The history command will show you every command you have recently, and maybe not-so-recently, executed. Each command will also have a number by it. You can type `!number` to then immediately re-execute that command. Additionally, you can provide a string instead of the number to the `!` command. This will immediately re-execute the last command executed that matches the given string, as illustrated below.

```
211 lesath.cs.pdx.edu> gcc -v
Reading specs from /pkgs/gcc/gcc-2.95.2/...
gcc version 2.95.2 19991024 (release)
212 lesath.cs.pdx.edu> gzip --version
gzip 1.2.4 (18 Aug 93)
Compilation options:
DIRENT UTIME STDC_HEADERS HAVE_UNISTD_H
213 lesath.cs.pdx.edu> !g
gzip --version
gzip 1.2.4 (18 Aug 93)
Compilation options:
DIRENT UTIME STDC_HEADERS HAVE_UNISTD_H
214 lesath.cs.pdx.edu> !gc
gcc -v
Reading specs from /pkgs/gcc/gcc-2.95.2/...
gcc version 2.95.2 19991024 (release)
215 lesath.cs.pdx.edu> !gz
gzip --version
gzip 1.2.4 (18 Aug 93)
Compilation options:
DIRENT UTIME STDC_HEADERS HAVE_UNISTD_H
```

4 Tab Completion

The last feature of the shell this tutorial will discuss is tab completion. You can type in only a portion of a command or filename, and then press *Tab*. The shell will then try and locate a unique match for a command or filename based on the portion you have provided. It will try to complete the name as far as possible. For example, if I type `cv` the shell will complete it to `cvs`. If I type in only the letter `u` the shell will not do any completion because several different commands begin with a `u`. However, the shell will complete it to `uncomp` because all commands that begin with “`unco`” also match `uncomp`. From there I could provide the shell with another character, say `r`, and try completion again. This would then match the command `uncompress`. Unfortunately, the power of tab completion is somewhat hard to describe in a text tutorial. The best way to become familiar with it is to try using it.

5 Conclusion

Having read this tutorial, you should now be able to work comfortably and ably with the shells and the Unix environment. To learn more about the basic Unix commands you should peruse the other tutorials we have available. If you wish to learn more about shells and customizing your environment, the tutorial entitled ‘Environment Customization’ is highly recommended, and it builds on the knowledge gained in this tutorial.