Unix Beginners QuickSheet

Version: 0.10.0
Date: 2/23/7

This document is written for those who find themselves new to the Unix command line (shell). Each Unix version has local variations in terms of availible shells, commands, and directory layout. This is intended as a general guide only, verify these commands exist and are compatible by refering to your local man page.
Unix is a popular operating system that is built some of the most mature developments in operating system design. Many of the basic concepts that are used today (and are covered in this guide) were designed over 30 years ago. An knowledge investment in Unix skills is likely to be a rewarding investment for a significant ammount of time. This gives the Unix user the oppurtunity to develop a deeper and more profound knowledge of their tool over the OS user who must re-learn the interface every major release.

Definitions

alias A *command* that is really another command or series of commands. The command `ll` is frequently an alias for `ls -l`. Whenever you type an alias (such as `ll`) the shell substitutes the actual command `ls -l`.

argument This is an additional parameter that follows the command on the command line that tells the command how you expect it to behave. The parameter `-l` to the command `ls` tells `ls` that you want a long listing of directory items instead of the standard shortend list that you would get using only `ls` by itself.

daemon This is a program that runs in the background and handles tasks and requests made to the system. Names of daemon processes frequently end in the letter "d" by convention. One example of this is the httpd process that handles http requests. (The httpd daemon is more comonly known as a web server.)

internal (command) Some commands are processed by the shell and not by the local system. One such example is the cd command. If `cd` were a binary it would launch in its own environment, change the directory and exit back to the original environment with the original directory. For this reason, commands that effect the environment, must be internal to the shell. These are also sometimes called builtins. These commands can be determined with the *type* shell internal command.

link A link is a pointer to a file. It is frequently used to give a file or directory multiple names. Links are either symbolic or hard. Symbolic links are actually files that point to other files. Symbolic links can refer to directories, work across filesystems, or refer to files that currently do not exist. Hard links are when a file has multiple different names. Use `ln` to create links.

pipe This is represented by the | character. When a pipe is placed between two commands it connects the output (*stdout*/*stderr*)of the first command to the input (*stdin*) of the second command. This is the primary method of creating larger functionality by *joining* multiple smaller commands.

process A program that has been read from disk and is running is called a process. Each process has a process identifier called a pid. A list of processes and pids can be viewed with the ps command.

stderr Standard error. This is the stream of data that programs create strictly for error messages. Frequently this is mixed with *stdout* data and may seem identical. For most purposes `stdout` and stedrr are identical, but the key difference is when used in long connections of commands the stderr can be isolated and sent to a different location if necessary. stderr is known as file descriptor "2".

stdin Standard in. This is the stream of data that a program recieves from the command line. Not to be confused with command line `arguments` that are specified when the program is run this is data that is recieved from user input or other commands while the program is running. Input can be sent from program to program by *piping* it between the two using ones stdin and anothers stdout.

stdout Standard out. This is the stream of data that (command line) programs print to the user interface. This output can be captured via a *pipe* or a *redirect* and sent to another program or file. stdout is know as file descriptor "1".

environment This is the collection of variables and settings that a program runs within. Each environment is inherited from the process that started it. Changes in your environment (such as setting a variable) may be inherited by processes you start, but will not have an effect on processes that are allready running. This is a key componet of a multi-user environment.

The shell

• The Unix shell is the primary method that users interact with the system. It is the essence of the command line interface experience with the system. In the context of this paper, the shell is the essence of the Unix experience. It is responsible for interpreting user requests entered on the command line and linking groups of commands together.
• The shell is the *interactive* interface to the system that is utilized when running locally or when connecting remotely over a telnet or ssh connection.
• The most common shells are bourne (sh), korn (ksh), C (csh), and bourne-again (bash). This document deals primarily with bourne and its descendents (ksh and bash).
• Unix is case sensitive. The file wfavorite.txt is not the same as WFavorite.txt. Unix views these as two different items.

Shell shortcuts - file globbing

• File globbing allows you list multiple files by expressing all the files as a pattern. The examples below use the echo to simply repeat the *expanded* glob.
• The * character can match any number of characters. By itself the * will match everything. The number of matches can be limited by using other characters with it. The ? character matches any single character. The ? by itself will match any single letter name.
Print all files in the current directory that end in txt
```
echo *txt
```
Print all files in the current directory that begin with "important"
```
echo important*
```
Print all files in the current directory that have four characters
```
echo ????
```
Print all files in the /usr/bin directory that begin with X
```
echo /usr/bin/X*
```

Shell advanced shortcuts

• shell history is the ability to call upon and reuse previously run commands in the shell. The history can be viewed with the `history` shell command, or can be reused with one of the command line key sequences. The previous command is typically available by using the up arrow key. If this does not work Ctrl-p (emacs mode) or Esc-k (vi mode) will recall the previous command.
• Most shells allow for a concept called command completion. This is often implemented as a Tab (in `bash` shell), Esc Esc (`korn` emacs mode), or Esc\ (`korn` vi mode) key sequence. Simply type the first string of characters in a filename and then hit the command completion sequence, the shell will complete the file name from the possibilites it finds.

Quirks of the shell

• When you type a command in Unix the shell will set about finding it. If you do not specify a path (such as `/usr/bin/ls`) but instead call the command without a path (such as `ls`) the shell is responsible for finding it. To do this the shell uses a list of directories that it stores in the `$PATH` variable. If the command you are trying to run is not in these directories it will not be found, even if it is in the directory you are in. (It is considered bad practice to put the current directory (a '.') in your `$PATH` to change this behavior.)
• The shell has *internal* and *external* commands. Internal commands are implemented by the shell itself. External commands are implemented by other programs that exist as seperate programs (executables) on the system. Commands like `ls` are external. In the case of ls this command tends to be in `/bin/ls`. The cd command is an internal shell command that is not found in the system but is interpreted by the shell itself as a request to change the current directory that it is working in.
• When a shell starts a command the relationship is called parent-child. The naming is appropriate because the children *inherit* the environment of the parent (shell). This means that any variables or settings that were set in the shell environment become the settings for the child. Changes made while in the child process do not change the environment of the parent. Variables that have been exported in the parent are visible in the child. For this reason some programs will require that a list of variables be set and exported prior to them running.
• When a command is run from the shell it runs in the foreground. This means that all input and output are used for that command. If a command is backgrounded it will run in the background and you can continue to run additional commands in the foreground. A command can be backgrounded by calling it followed with a `&` character. Commands that *require* input should not be backgrounded.

Understanding file attributes

• All files belong to someone, and have descriptions of what kind of files they are and how they can be used. This data is viewable with the `ls` command. Below is a sample listing created by running `ls -l`:
```
-rw-rw-r-- 1 wfavorite devlopers 62896 Oct 4 08:26 hw.c
-rwxrwxr-x 1 wfavorite devlopers 4679 Oct 16 08:21 hw
drwxrwxr-x 11 wfavorite devlopers 4096 Oct 15 08:15 work
```
From this we can determine that hw.c:
`-rw-rw-r--` is a file
`-rw-rw-r--` the owner can read, write, but not execute it
`-rw-rw-r--` the group can read, write, but not execute it
`-rw-rw-r--` all others can only read it
hw:
`-rwxrwxr-x` is also a file
`-rwxrwxr-x` both the user and group have read, write and execute
`-rwxrwxr-x` others can only read, and execute it
work:
`drwxrwxr-x` is a directory
`drwxrwxr-x` shares the same permisssions as hw. Because a directory cannot be executed the execute permission means that users can enter the directory and view its contents.
All of the above entries:
are owned by user wfavorite and group devlopers.
• File permissions are modified using the chmod command
• File ownership is modified using the chown and chrgp commands

Useful Unix pointers

• **Don't use spaces or strange characters in filenames.** Because you have to refer to files on the command line it is best to not use spaces in the file name. If you do use spaces it will be necessary to "quote" the name so that it does not appear as two different files when specified on the command line.

Commands (grouped by task)

Commands to modify file attributes
chmod chown
Commands to show info about files
file ls
Move, copy, or rename files
mv cp ln cpio
Show contents of a text file
more cat head tail
Show what processes are running on a system
ps top

Remote access

• Remote access to Unix systems is typically gained in one of the following ways: remote shell, file transfer, application access, or remote console / X.
• Remote shell is achived primarily by telnet or now, using the more secure method, ssh.
• Files are transfered *individually* by using ftp and rcp or the secure equivelants sftp and scp.

Commands listed with examples

cat concatenate and print files
Dump /etc/hosts file to stdout
`cat /etc/hosts`
Write everything typed (until Ctrl-d is entered) to the file "newfile"
`cat >> newfile`
cd Change (working) directories
Change to the /usr/bin directory
`cd /usr/bin`
Change to your home directory
`cd`
Change back to the directory you were previously in
`cd -`
cp Copy a file or directory
Copy "myfile" to "myfile.backup"
`cp myfile myfile.backup`
Copy the directory "mydir" to "mydir.backup"
`cp -r mydir mydir.backup`
cpio Copy a file or directory
Note: Options to copi vary by system, check your local options.
Copy "sourcedir" to "destdir"
`cd sourcedir; find . -print -depth | cpio -pdm destdir`
date Print the current date and time
exit Log out from the current session
Ctrl-d also works for this purpose
file Determine what kind of file a target is
Determine what kind of file "myfile" is
`file myfile`
Determine what kind of file the (special) file "/dev/zero" is
`file /dev/zero`
find Locate a file in the system
Find a file "myfile" in your home directory, print all matching names
`find ~/ -name myfile -print`
grep Search through the contents of a file (or stream) for a pattern
Look for a line with the word needle in the file called haystack.txt
`grep needle haystack.txt`
Look for a line that contains *only* needle in haystack.txt
`grep "^needle$" haystack.txt`
Look for the string "emacs" (a running process) in the output of ps
`ps | grep emacs`
head Print the first part of a file
Print the first (by default) 10 lines of "myfile.txt"
`head myfile.txt`
id Display current user and group membership
ls List all files in a directory
List file names
`ls`
List names and attributes of files
`ls -l`
man View the online manual
View the manual page for the ls command
`man ls`
mkdir Create a directory
Create a directory called mydir in the current working directory
`mkdir mydir`
more List the contents of a file (or stream of data) a bit at a time
Display the contents of a file called "largefile"
`more largefile`
Display the results of the ps command a page at a time
`ps | more`
mv Move a file to a new location or name
Move "myfile" into the directory called "mydir"
`mv myfile mydir`
Rename "myfile" to "ourfile"
`mv myfile ourfile`
passwd Change your password on the system
ps List running proceses
List processes that you have running
`ps`
List processes that belong to all users
`ps ax` (for OS X) `ps -ef` (for Linux, Solaris)
pwd Print the current working directory
rm Remove a file or directory
Remove a file called "myfile"
`rm myfile`
Remove a direcotry called "mydir"
`rm -r mydir`
su Become another user
Simply become the user fred (gain fred's permissions)
`su fred`
Become the user fred and run the login profile for that user
`su - fred`
Become the root user and run the profile (user environment) for root
`su -`
sudo Run a command as another user (typically root)
Edit the system hosts file
`sudo vi /etc/hosts`
Run the id command as root (worthless, but demonstrates sudo)
`sudo id`
tail Print the last part of a file
Print the last (by default) 10 lines of "myfile.txt"
`tail myfile.txt`
top List the top (CPU) processes on a system
type Determine *what* a string / label is
Determine if cd is a binary, alias, or internal command.
`type cd` (cd is an internal command / shell builtin)
which Search your path for a binary
Show the full path to the ls command
`which ls`

Piping and redirection

• Piping is typically done between programs while redirection is typically done to or from files. Some examples are listed here.
To pipe the stdout and stderr of ls to sort
`ls | sort`
Send the stdout of ls to sort and the stderr to the file /dev/null
`ls 2> /dev/null | sort`
Send both the stdout and stderr of ls to the file /dev/null
`ls > /dev/null 2>&1`
Overwrite the contents of `file.txt` with the output of `ls`
`ls > file.txt`
Append the output of ls to the end of file.txt
`ls >> file.txt`

Editors - Using `vi`

• More user friendly editors exist for Unix systems, but the `vi` editor can be found on virtually every Unix system you may ever encounter.
• `vi` has a command mode and an insert mode. Command mode is for telling `vi` things you want to do, such as open, save, edit a file, or quit. Insert mode is when you are actually typing text into the file.
• Some essential command mode commands:
:w Enter - Write the file
:q Enter - Quit the editor
:q! Enter - Quit the editor without saving changes you may have made
i - Begin inserting text where the cursor is in the file
x - Delete the character under the cursor
dd - Delete entire line
p - Paste previously deleted line
Esc - (While in insert mode) will return you to command mode
• The commands for `vi` and other editors are very *rich*. The full set of commands is beyond the scope of this QuickSheet.

Moving around the system - directories

Directories are delimited with a / character.
`.` referes to the current directory
`..` refers to the parent directory
`cd ..` will move you to the parent directory
`cd -` will change to the previous directory
`pwd` will print the current directory

Troubleshooting

• What shell am I running?
`echo $0` (echo $SHELL is not necessarily the one you are *running*)
• Why does my Backspace key print funny chars (like ^H)?
`stty erase ^H` ("stty erase " then backspace key to define the backspace key)
• Why does it say "file not found" when I run `./ls`?
Because you said to run the copy of ls in the current directory and it does not exist there. Try: `ls` (If it does not work, you most likely have a problem with your PATH.
• Why can I only create a file 1 (or 2) gig in size?
Your account may have limits set. Check your ulimit for files using the `ulimit` command.

Special files

• Some "files" on the system have special purposes. These files are used by scripts and programs to generate or remove data. Some examples are:
`/dev/null` - Anything written to this file will disappear
`/dev/zero` - Anything read from this file will be all (binary) zeros
`/dev/random` - Anything read from this file will be random data

Additional help

Find all instances of "system" in the man files
`whatis system` -or- `apropos system`
Access the man(ual) page for the man browser
`man man`

About this QuickSheet

Created by: William Favorite (wfavorite@tablespace.net)
Updates at: http://www.tablespace.net