

UNIX Tips and Tricks

Mark Howison

User Services & Support

Other upcoming workshops

- ▶ “Profiling and Performance Analysis”
 - Tuesday, March 15, 2-3pm, Saloman 003
- ▶ “Parallel I/O Libraries and Techniques”
 - Monday, April 4, 1-2pm, Petteruti Lounge
- ▶ We will probably repeat this semester's workshop schedule every semester
- ▶ We may also plan a multi-day “boot camp” in the summer, covering the same topics
- ▶ Please let us know if you have specific requests for other topics!

Command-line interfaces

- ▶ All CCV systems run a UNIX-like operating system called Linux
- ▶ You can interact with a UNIX system through a *command-line interface*
 - Type and enter commands at a *prompt*, e.g.
`[ccvuser@login001 ~]%`
 - Want to know more about a command? Look at its manual or “man” page using
`[ccvuser@login001 ~]% man commandname`
 - OR most commands have a “help” message that you can print with
`[ccvuser@login001 ~]% commandname -h`
`[ccvuser@login001 ~]% commandname --help`

Logging into a UNIX system

- ▶ Most UNIX systems use Secure Shell (SSH) for remote logins
 - Mac OS X and Linux come with a command called “ssh” that you can run from the terminal
 - To connect to the Oscar system:
`% ssh username@ssh.ccv.brown.edu`
 - Windows clients are also available: try PuTTY or Cygwin

Shells

- ▶ The command line interface is controlled by a program called a “shell”
- ▶ There are several different shells available, each with slight differences
- ▶ This talk assume the `bash` shell (default on most Linux distributions and Mac OS X)
- ▶ New CCV accounts defaulted to `csh` until recently
- ▶ To change to `bash`:
[ccvuser@login001 ~]% `ypchsh`
(enter password)
(enter `/bin/bash`)

Directories

- ▶ View the current directory with the “pwd” (print working directory) command:

```
[ccvuser@login001 ~]% pwd  
/users/ccvuser
```
- ▶ Change directories with the “cd” command:

```
[ccvuser@login001 ~]% cd scratch  
[ccvuser@login001 scratch]% pwd  
/users/ccvuser/scratch
```
- ▶ Make new directories with the “mkdir” command:

```
[ccvuser@login001 scratch]% mkdir testdir  
[ccvuser@login001 scratch]% cd testdir  
[ccvuser@login001 testdir]% pwd  
/users/ccvuser/scratch/testdir
```

Directories (Cont'd)

- ▶ List the contents of a directory with the “ls” command:

```
[ccvuser@login001 ~]% ls  
batch.script data README scratch
```

- ▶ “-l” for long list (with more details)
- ▶ “-a” for all files (including those that start with “.”)
- ▶ “-rt” lists most recent files last
- ▶ “-h” prints human-readable file sizes (e.g. in KB, MB, etc. instead of bytes)

Special directories

- ▶ ~ is your home directory (also \$HOME):
 - That is, you can change to your home directory with:
% cd ~
- ▶ . is the current directory
- ▶ .. is the parent directory
- ▶ / is the root directory
- ▶ Move back to the previous working directory with:
% cd -

Moving files

- ▶ Move (rename) a single file with:

```
% mv file newfilename
```

- ▶ Move multiple files to a different directory:

```
% mv file1 file2 file3 file10 newdir/
```

- ▶ Or use wildcards:

- * matches any number of any characters

```
mv file* newdir/ (matches all the files above)
```

- ? matches one of any character

```
mv file? newdir/ (matches all the files except file10)
```

- ▶ “-i” = interactive mode: prompts before overwriting existing files

Copying files

- ▶ Copy a single file with:
`% cp file duplicatefile`
- ▶ Recursively copy entire directories (and subtree) with “-r”:
`% cp -r dir newdir`
- ▶ Copy only files that will not overwrite existing files with “-n”

Copying files over SSH

- ▶ Use Secure Copy (scp):

```
% scp file username@host:/path
```
- ▶ To copy an entire directory recursively, use “-r”

Links (e.g. aliases or shortcuts)

- ▶ Create another name for a file or directory without copying it
- ▶ Two types: *symbolic* and *hard*
- ▶ More typical to use symbolic:

```
% ln -s sourcefile newalias
```
- ▶ Show up as → in file list:

```
[ccvuser@login001 ~]% ls -al  
lrwxrwxrwx  1 ccvuser ccvgroup 27 Sep 7 12:52 batch.script  
                -> /gpfs/home/doc/batch.script
```

Auto-completion

- ▶ In commands that require you to input a path, you don't have to type the entire path!
- ▶ Hit tab to list all possible completions of the path you've typed so far
 - If there is only one possible complete, bash will fill it in for you

Viewing files

- ▶ `cat`: “concatenate” contents of a file to the display
- ▶ `more`: display contents of a file one page at a time
 - Can also use `less`
- ▶ Text editors on the command line:
 - `vi` or `emacs`: powerful, steep learning curve
 - `pico`: easier to use, less powerful
- ▶ Alternative to a command line editor:
 - Mount your Oscar directories through Samba/CIFS and use a GUI text editor on your local machine
 - Use the VNC client with `gedit` or similar GUI editor

Size of files and directories

- ▶ Sizes of individual files is displayed by `ls -l`
- ▶ Find sizes (**disk usage**) of entire directories with:
`% du dir/`
- ▶ Lists bytes for every file in directory (recursively)
- ▶ For **human-friendly** output and **sum-only**:
`% du -sh dir/`

File ownership

- ▶ Every file and directory is associated with both a user and a group
- ▶ **Change ownership** of a file with:
`% chown user:group file`
- ▶ Recursively change ownership of an entire directory (and subtree) with “-R”:
`% chown -R user:group dir/`
- ▶ To only change the group, use `chgrp`
- ▶ View your group membership with `groups`
 - Or others' with `groups username`

File permissions

- ▶ Every file and directory has *read*, *write*, and *execute* permissions for its *user*, *group*, and *other* (everyone else).
- ▶ **Change the permissions (or **mode**) of a file with:**

```
% chmod [augo][+-][rwx] file
```
- ▶ **Examples:**

```
% chmod a+r file      (all = user, group, and other, can read)  
% chmod g+w file      (group can write)  
% chmod a+rwx file    (all can read, write, execute)  
% chmod o-rwx file    (others can't read, write, execute)
```
- ▶ **Use “-R” for the entire contents of a directory**

```
% chmod -R g+w dir/
```

Redirecting output

- ▶ Redirect to a file (and *truncate* it) using `>`
`% ls -la >filelist`
- ▶ To also capture errors, use `>&`
- ▶ Use `>>` to *append* instead of *truncate*
- ▶ Redirect the output of one command to the input of another command using a *pipe*: `|`
`% ls -la | more`
 - Especially useful with `more` to page through long outputs
 - Also useful with `grep`...

Pattern matching

- ▶ `grep`: find a pattern in a file or command output

```
% grep pattern file
% commandname | grep pattern
```
- ▶ Pattern is a “regular expression”
 - Very flexible, but can be complicated to construct
 - Certain characters (`.` `*` `+` `?`) are “special”
 - Simplest case: just use plain text, like `grep "error"`
- ▶ “-o” to only print the exact match (not the entire line)
- ▶ “-P” to use Perl-style regular expressions
- ▶ “-c” to count the number of matches
- ▶ “-C n” to print n lines around the match

Command history

- ▶ List your history of previous commands with:
`% history`
- ▶ Search for previous commands by piping to grep:
`% history | grep commandname`
- ▶ Execute a previous command with:
`% !number`
- ▶ Execute the last command matching a pattern:
`% !pattern`
- ▶ Print but do not execute a previous command:
`% !number:p`

Environment variables

- ▶ You can set and access string variables in `bash`:
% `VARNAME=value`
% `echo $VARNAME`
(only set for your current session)
- ▶ Convention is to use all caps
- ▶ Many variables are set by default. View all of your variables with:
% `env`

Command aliases

- ▶ An *alias* changes the way a command is interpreted by the shell

- ▶ List your current aliases with:

```
% alias
```

- ▶ Create a new alias with:

```
% alias newcommand='command args'  
(this only stays set for your current session)
```

- ▶ Examples:

```
% alias ll='ls -la'  
% alias cp='cp -i'  
% alias rm='rm -i'
```

Initialization files

- ▶ Stores variables and aliases that you want to load everytime you login
- ▶ `~/.bash_profile` is read by *interactive login* shells (e.g. when you log in through SSH)
- ▶ `~/.bashrc` is read by *interactive non-login* shells
- ▶ When you launch a new shell from your current shell, it inherits the current environment

Batch scripts

- ▶ Sometimes you want to automate frequently used commands or workflows
- ▶ You can write a batch of commands in a text file called a *script*, then execute them all at once with:

```
% bash scriptname
```
- ▶ Or you can place at the top of the script:

```
#!/bin/bash
```

and change the permissions of the script to +x to execute it directly:

```
% ./scriptname
```

Batch scripts (Cont'd)

- ▶ Scripts can take arguments, and you can access them with the special variables \$1, \$2, etc.
- ▶ You can set variables temporarily within the script
- ▶ To modify the environment of the shell calling the script, you can “source” the script for “exported” variables
 - If your script has:
`export PATH=~ /bin:$PATH`
and you source it with
`% source scriptname`
your environment will now have a new value for PATH

Job control

- ▶ Add & to the end of a command to run it in the *background*
- ▶ Or, if a command is already running, use Ctrl-Z (stop) then the bg command
- ▶ To resume the last backgrounded job, use fg
- ▶ Use jobs to list all running jobs, and bg %N or fg %N to background or foreground job N

Shortcuts

- ▶ Ctrl-A: go to the beginning
- ▶ Ctrl-E: go to the end
- ▶ Ctrl-<Arrow>: move forward or back one word
- ▶ Ctrl-W: delete last word
- ▶ Ctrl-U: delete up to cursor
- ▶ Ctrl-K: delete after cursor
- ▶ Ctrl-C: interrupt
- ▶ Ctrl-R: search through command history
- ▶ <Up>: show last command
- ▶ Ctrl-D: logout

Using SSH keys

- ▶ Tired of typing your password everytime you login?
- ▶ You can create a “key” file that identifies you.
 - This key can also be protected by a “passphrase” that you type once when you load the key
 - Create the key on your local system with

```
% ssh-keygen -t dsa
```
 - This will create both *public* and *private* key files called `id_dsa.pub` and `id_dsa` in `~/.ssh`
 - Don't share `id_dsa` with anyone!
 - Place `id_dsa.pub` on the remote system and add it to your authorized key list on the remote system with

```
% cat id_dsa.pub >> ~/.ssh/authorized_keys
```

Using SSH keys (Cont'd)

- ▶ On your local machine, load your private key with

```
% ssh-add ~/.ssh/id_dsa
```
- ▶ You will enter your key's passphrase once
 - Everytime you restart your computer, you will have to enter the passphrase again
- ▶ Now you can connect to the remote machine without entering a password