Unix Systems Programming

Course:	CSC-507
Professor:	Charles E Campbell
Website:	http://www.drchip.org/astronaut/cua/csc507/
Email:	charles.e.campbell@nasa.gov

Catholic University of America

Unix Tools

Topics: shells, basic compiling, make, gdb, and debugging

Basic Compiling

- **cc** standard name for C compilers
- **CC, cxx** usual names for C++ compilers
- **gcc, g++** Gnu C/C++ compilers
- **cc filename.c -o outfile** : compile and link
- **cc -c filename.c** compile to object (.o)
- **cc -O filename.c -o outfile** compile and optimize
- **cc -g filename.c -o outfile** include symbolic tables (good with gdb)

C++ compilers usually have these same options, at least under Unix/Linux.

Windows compilers are idiosyncratic.

Shell Scripts: Initialization

- Unix command-line interfaces come in several flavors.

Shell	***Login Shell Initialization File***	***Interactive Shell Initialization File***
bash	.profile, .bash_profile	.bashrc
csh	.login	.cshrc, .tcshrc
ksh	.profile	.kshrc (or as specified by $ENV)
sh	.profile	N/A
tcsh	.login	.tcshrc
zsh	.zlogin	.zshrc

Shell Scripts: Initialization

- Unix command-line interfaces come in several flavors.

Shell	***Login Shell Initialization File***	***Interactive Shell Initialization File***
bash	.profile, .bash_profile	.bashrc
csh	.login	.cshrc, .tcshrc
ksh	.profile	.kshrc (or as specified by $ENV)
sh	.profile	N/A
tcsh	.login	.tcshrc
zsh	.zlogin	.zshrc

- The interactive shell initialization files are executed before any shell script to initialize it

Shell Scripts: Initialization

- Unix command-line interfaces come in several flavors.

Shell	***Login Shell Initialization File***	***Interactive Shell Initialization File***
bash	.profile, .bash_profile	.bashrc
csh	.login	.cshrc, .tcshrc
ksh	.profile	.kshrc (or as specified by $ENV)
sh	.profile	N/A
tcsh	.login	.tcshrc
zsh	.zlogin	.zshrc

- The interactive shell initialization files are executed before any shell script to initialize it
- The login shell initialization files are executed only for “login” sessions

Shell Scripts: Initialization

- Unix command-line interfaces come in several flavors.

Shell	***Login Shell Initialization File***	***Interactive Shell Initialization File***
bash	.profile, .bash_profile	.bashrc
csh	.login	.cshrc, .tcshrc
ksh	.profile	.kshrc (or as specified by $ENV)
sh	.profile	N/A
tcsh	.login	.tcshrc
zsh	.zlogin	.zshrc

- The interactive shell initialization files are executed before any shell script to initialize it
- The login shell initialization files are executed only for “login” sessions
- One may put any number of commands in script files

Shell Scripts: Initialization

- Unix command-line interfaces come in several flavors.

Shell	***Login Shell Initialization File***	***Interactive Shell Initialization File***
bash	.profile, .bash_profile	.bashrc
csh	.login	.cshrc, .tcshrc
ksh	.profile	.kshrc (or as specified by $ENV)
sh	.profile	N/A
tcsh	.login	.tcshrc
zsh	.zlogin	.zshrc

- The interactive shell initialization files are executed before any shell script to initialize it
- The login shell initialization files are executed only for “login” sessions
- One may put any number of commands in script files
- One chooses a shell when the account it made; the system administrator is responsible for this.

Quick Overview

- `/` The root directory

  Example 1 `cd /` *(change directory to the root directory)*

Quick Overview

`/` The root directory

Example 1 `cd /` *(change directory to the root directory)*

filenames every directory name or file name can be any sequence of characters other than “/”. Case matters!

Example 2 `cat somefile` *(dump contents of somefile to display)*

Example 3 `cd /home/somedir` *(change directory to specified directory)*

Quick Overview

`/` The root directory

Example 1 `cd /` *(change directory to the root directory)*

filenames every directory name or file name can be any sequence of characters other than “/”. Case matters!

Example 2 `cat somefile` *(dump contents of somefile to display)*

Example 3 `cd /home/somedir` *(change directory to specified directory)*

`.` shorthand for current directory

Example 4 `ls .` *(list files in currrent directory)*

Quick Overview

`/` The root directory

Example 1 `cd /` *(change directory to the root directory)*

filenames every directory name or file name can be any sequence of characters other than “/”. Case matters!

Example 2 `cat somefile` *(dump contents of somefile to display)*

Example 3 `cd /home/somedir` *(change directory to specified directory)*

. shorthand for current directory

Example 4 `ls .` *(list files in currrent directory)*

.. containing directory

Example 5 `ls ..` *(list files in directory above current one)*

Quick Overview, con’t.

file permissions consist of four categories, each with its own octal number

1st	setuid/setgid (special permissions)
2nd	owner
3rd	group
4th	others

Example 6 *Enter:* `ls -lsa somefile`

Result: `0 -rw-------. 1 cec cec 0 2011-12-29 18:32 somefile`

(list detailed information on “somefile”)

Quick Overview, con't.

permissions Add the following sets of numbers to come up with the desired permission for each category

read	4
write	2
execute	1

Example 7 `chmod 644 myfile`

(change permissions to read-write for myself and read-only for my group and others)

$$\begin{array}{lcl} 6 & = & 4+2\text{: user has read and write} \\ 4 & = & 4\text{: group has read only} \\ 4 & = & 4\text{: others have read only} \end{array}$$

Quick Overview, con't.

permissions Add the following sets of numbers to come up with the desired permission for each category

read	4
write	2
execute	1

Example 7 `chmod 644 myfile`

(change permissions to read-write for myself and read-only for my group and others)

$$\begin{aligned} 6 &= 4+2\text{: user has read and write} \\ 4 &= 4\text{: group has read only} \\ 4 &= 4\text{: others have read only} \end{aligned}$$

standard file descriptors control terminal-based input and output streams

0	stdin	standard input
1	stdout	standard output
2	stderr	standard error (output)

Executing Simple Commands

At the command prompt, typing the command name, any arguments, and then pressing $\boxed{\text{return}}$ will cause that command to execute.

Example 8 *Enter:* `date`

Result: *Thu Dec 29 18:37:50 EST 2011*

Executing Simple Commands

At the command prompt, typing the command name, any arguments, and then pressing $\boxed{\text{return}}$ will cause that command to execute.

Example 8 *Enter:* `date`

Result: *Thu Dec 29 18:37:50 EST 2011*

Example 9 *Enter:* `echo "hello"`

Result: *hello*

Executing Simple Commands

At the command prompt, typing the command name, any arguments, and then pressing $\boxed{\text{return}}$ will cause that command to execute.

Example 8 *Enter:* `date`

Result: *Thu Dec 29 18:37:50 EST 2011*

Example 9 *Enter:* `echo "hello"`

Result: *hello*

Example 10 `printenv`

Result: *(list of environment variables and their values)*

Executing Simple Commands

At the command prompt, typing the command name, any arguments, and then pressing $\boxed{\text{return}}$ will cause that command to execute.

Example 8 *Enter:* `date`

Result: *Thu Dec 29 18:37:50 EST 2011*

Example 9 *Enter:* `echo "hello"`

Result: *hello*

Example 10 `printenv`

Result: *(list of environment variables and their values)*

Example 11 `man ls`

Result: *(ls lists files in current directory; man provides manual pages)*

Executing Simple Commands

At the command prompt, typing the command name, any arguments, and then pressing $\boxed{\text{return}}$ will cause that command to execute.

Example 8 *Enter:* `date`

Result: Thu Dec 29 18:37:50 EST 2011

Example 9 *Enter:* `echo "hello"`

Result: hello

Example 10 `printenv`

Result: (list of environment variables and their values)

Example 11 `man ls`

Result: (ls lists files in current directory; man provides manual pages)

Example 12 `pwd`

Result: (display name of current directory)

Editing Old Commands (Command History)

Enter: `export EDITOR=vi`

`esc`

Then vi-style editing is enabled.

history: this command will display recent commands executed by the shell

Editing Old Commands (Command History)

Enter: `export EDITOR=vi`
`esc`

Then vi-style editing is enabled.

history: this command will display recent commands executed by the shell

A Sampling of Editing Commands:

h go left in command

Editing Old Commands (Command History)

Enter: `export EDITOR=vi`
$\boxed{\texttt{esc}}$

Then vi-style editing is enabled.

history: this command will display recent commands executed by the shell

A Sampling of Editing Commands:

h go left in command

j go down history list

Editing Old Commands (Command History)

Enter: `export EDITOR=vi`

$\boxed{\texttt{esc}}$

Then vi-style editing is enabled.

history: this command will display recent commands executed by the shell

A Sampling of Editing Commands:

h go left in command

j go down history list

k go up history list

Editing Old Commands (Command History)

Enter: `export EDITOR=vi`

$\boxed{\text{esc}}$

Then vi-style editing is enabled.

history: this command will display recent commands executed by the shell

A Sampling of Editing Commands:

h go left in command

j go down history list

k go up history list

l go right in command

Editing Old Commands (Command History)

Enter: `export EDITOR=vi`

$\boxed{\texttt{esc}}$

Then vi-style editing is enabled.

history: this command will display recent commands executed by the shell

A Sampling of Editing Commands:

h go left in command

j go down history list

k go up history list

l go right in command

/pat search for command pattern

Editing Old Commands (Command History)

Enter: `export EDITOR=vi`

$\boxed{\text{esc}}$

Then vi-style editing is enabled.

history: this command will display recent commands executed by the shell

A Sampling of Editing Commands:

h go left in command

j go down history list

k go up history list

l go right in command

/pat search for command pattern

x delete character under cursor

Editing Old Commands (Command History)

Enter: `export EDITOR=vi`

$\boxed{\texttt{esc}}$

Then vi-style editing is enabled.

history: this command will display recent commands executed by the shell

A Sampling of Editing Commands:

h	go left in command
j	go down history list
k	go up history list
l	go right in command
/pat	search for command pattern
x	delete character under cursor
rc	replace character under cursor with “c”

Editing Old Commands (Command History)

Enter: `export EDITOR=vi`

$\boxed{\texttt{esc}}$

Then vi-style editing is enabled.

history: this command will display recent commands executed by the shell

A Sampling of Editing Commands:

h go left in command

j go down history list

k go up history list

l go right in command

/pat search for command pattern

x delete character under cursor

rc replace character under cursor with “c”

D delete from cursor to end-of-command

Aliases

Aliases shorten typing needs:

`alias word="command + args`

Whenever “word” is encountered, the shell will see “command+args”.

Careful: no spaces between “word” and `"`!

Example 13 `alias cp='/bin/cp -i'`

(copy files, but ask before overwriting)

Aliases

Aliases shorten typing needs:

`alias word="command + args`

Whenever “word” is encountered, the shell will see “command+args”.

Careful: no spaces between “word” and `"`!

Example 15 `alias cp='/bin/cp -i'`

(copy files, but ask before overwriting)

Pipes

Commands’ output can be fed into the input of another command. Commands run concurrently.

Example 16 `echo "who | fgrep cec"`

(list current users on the computer; select those strings having “cec”)

I/O Redirection

>file	stdout	output to file
>>file	stdout	appends to file
<file	stdin	use file contents as input
2>file	stderr	error output to file
2>>file	stderr	error output appended to file
2>&1	stderr	error output redirected to stdout

(">&" means "redirect a copy of")

I/O Redirection

>file	stdout	output to file
>>file	stdout	appends to file
<file	stdin	use file contents as input
2>file	stderr	error output to file
2>>file	stderr	error output appended to file
2>&1	stderr	error output redirected to stdout

(">&" means "redirect a copy of")

Example 17 `echo "this is a string" > afile`

I/O Redirection

>file	stdout	output to file
>>file	stdout	appends to file
<file	stdin	use file contents as input
2>file	stderr	error output to file
2>>file	stderr	error output appended to file
2>&1	stderr	error output redirected to stdout

(">&" means "redirect a copy of")

Example 17 `echo "this is a string" > afile`

Example 18 `echo "this is a string, too" >> afile`

I/O Redirection

>file	stdout	output to file
>>file	stdout	appends to file
<file	stdin	use file contents as input
2>file	stderr	error output to file
2>>file	stderr	error output appended to file
2>&1	stderr	error output redirected to stdout

(">&" means "redirect a copy of")

Example 17 `echo "this is a string" > afile`

Example 18 `echo "this is a string, too" >> afile`

Example 19 `fgrep string < afile`

I/O Redirection

>file	stdout	output to file
>>file	stdout	appends to file
<file	stdin	use file contents as input
2>file	stderr	error output to file
2>>file	stderr	error output appended to file
2>&1	stderr	error output redirected to stdout

(">&" means "redirect a copy of")

Example 17 `echo "this is a string" > afile`

Example 18 `echo "this is a string, too" >> afile`

Example 19 `fgrep string < afile`

Example 20 `cc myprog.c -o myprog 2> tmp`

Pathname Expansion

~	stands for your home directory
~cec	stands for “cec”’s home directory
`*`	matches any string of characters
`?`	matches any single character
`[···]`	matches any character(s) in the braces.
	Also handles ranges such as `[a-zA-Z]`

Pathname Expansion

~	stands for your home directory
~cec	stands for “cec”’s home directory
`*`	matches any string of characters
`?`	matches any single character
`[···]`	matches any character(s) in the braces.
	Also handles ranges such as `[a-zA-Z]`

Example 21 *ls ~cec*

Pathname Expansion

~	stands for your home directory
~cec	stands for “cec”’s home directory
`*`	matches any string of characters
`?`	matches any single character
`[···]`	matches any character(s) in the braces.
	Also handles ranges such as `[a-zA-Z]`

Example 21 *ls ~cec*

Example 22 *fgrep word *.c*

Pathname Expansion

~	stands for your home directory
~cec	stands for "cec"'s home directory
`*`	matches any string of characters
`?`	matches any single character
`[···]`	matches any character(s) in the braces.
	Also handles ranges such as `[a-zA-Z]`

Example 21 *ls ~cec*

Example 22 *fgrep word *.c*

Example 23 *fgrep word *.[ch]*

Quoting

'. . . '	avoids any expansion of characters inside the single quotes
". . . "	double quotes remove magic, except from $
`. . . `	will run the command in the backquotes

Quoting

`'...'`	avoids any expansion of characters inside the single quotes
`"..."`	double quotes remove magic, except from $
`` `...` ``	will run the command in the backquotes

Example 24 `echo '$USER'` *(shows the string “$USER”, not its value)*

Quoting

`'...'`	avoids any expansion of characters inside the single quotes
`"..."`	double quotes remove magic, except from $
`` `...` ``	will run the command in the backquotes

Example 24 `echo '$USER'` *(shows the string “$USER”, not its value)*

Example 25 `echo "$USER"` *(shows your userid)*

Quoting

`'...'`	avoids any expansion of characters inside the single quotes
`"..."`	double quotes remove magic, except from $
`` `...` ``	will run the command in the backquotes

Example 24 `echo '$USER'` *(shows the string “$USER”, not its value)*

Example 25 `echo "$USER"` *(shows your userid)*

Example 26 ``x=`ls $HOME` `` *(lists your home directory and saves it in a variable)*

Path/Command Execution

- If the command contains a `/`, ksh will attempt to execute it.

Path/Command Execution

- If the command contains a `/`, ksh will attempt to execute it.
- If the command is a reserved word, ksh follows its syntax.

  (ex. if-then-else-fi)

Path/Command Execution

- If the command contains a `/`, ksh will attempt to execute it.
- If the command is a reserved word, ksh follows its syntax.

  (ex. if-then-else-fi)
- If a built-in command, ksh executes it.

  (ex. alias)

Path/Command Execution

- If the command contains a `/`, ksh will attempt to execute it.
- If the command is a reserved word, ksh follows its syntax.

  (ex. if-then-else-fi)
- If a built-in command, ksh executes it.

  (ex. alias)
- Function: ksh executes the function in the current environment.

Path/Command Execution

- If the command contains a `/`, ksh will attempt to execute it.
- If the command is a reserved word, ksh follows its syntax.

  (ex. if-then-else-fi)
- If a built-in command, ksh executes it.

  (ex. alias)
- Function: ksh executes the function in the current environment.
- Alias: ksh substitutes the alias' text and executes it.

Path/Command Execution

- If the command contains a `/`, ksh will attempt to execute it.
- If the command is a reserved word, ksh follows its syntax.

  (ex. if-then-else-fi)
- If a built-in command, ksh executes it.

  (ex. alias)
- Function: ksh executes the function in the current environment.
- Alias: ksh substitutes the alias' text and executes it.
- PATH: ksh uses the PATH environment variable to search a list of directories for the command. Directories in the PATH are separated by colons. (:)

Command Substitution

To make the output of a command go into a variable or to be part of a string, use

$(· · ·)

(old style uses `· · · `)

Example 27 `x=$(ls)` *(list current directory and put into variable "x")*

Command Substitution

To make the output of a command go into a variable or to be part of a string, use

`$(···)`

(old style uses `'···'`)

Example 27 `x=$(ls)` *(list current directory and put into variable "x")*

Customizing Your Environment

- Whenever you login, ksh will execute the commands found in `$HOME/.profile`
- Whenever ksh executes something, it checks for an environment variable named `ENV`; if it exists, then ksh will first execute a script as named by `$ENV`.

An Example .profile

```
umask 077 % rwx------
set -o vi % use vi-style editing
set noclobber
PATH=".:$PATH"
export PS1='<esc>[m<esc>[1m${PWD##*/} <esc>[36m${HOSTNAME}?<esc>[m '
```

Environment Variables

environment variable A dynamic named set of variables with associated strings. These can effect how processes operate. Although subshells typically inherit the parent process' environment variables, child processes may add additional environment variables and may delete them.

Environment Variables

environment variable A dynamic named set of variables with associated strings. These can effect how processes operate. Although subshells typically inherit the parent process' environment variables, child processes may add additional environment variables and may delete them.

printenv This shell command will display the current set of environment variables.

Environment Variables

environment variable A dynamic named set of variables with associated strings. These can effect how processes operate. Although subshells typically inherit the parent process' environment variables, child processes may add additional environment variables and may delete them.

printenv This shell command will display the current set of environment variables.

	init	*accessing*	*accessing*	
creation	export x="abc"	$x	echo ${x}	(normal string)
	integer i i=2	$i	${i}	(integer)

(note: for bash, you'll want to use `alias integer='typeset -i'` first)

Environment Variables

environment variable A dynamic named set of variables with associated strings. These can effect how processes operate. Although subshells typically inherit the parent process' environment variables, child processes may add additional environment variables and may delete them.

printenv This shell command will display the current set of environment variables.

creation

init	*accessing*	*accessing*	
`export x="abc"`	$x	echo ${x}	(normal string)
`integer i` `i=2`	$i	${i}	(integer)

(note: for bash, you'll want to use `alias integer='typeset -i'` first)

deletion `unset envvar`

Comparisons and Expression Primitives

string1 = string2	compare two strings for equality

Comparisons and Expression Primitives

string1 = string2	compare two strings for equality
string1 != string2	compare two strings: not equal

Comparisons and Expression Primitives

string1 = string2	compare two strings for equality
string1 != string2	compare two strings: not equal
[[$\cdots$]]	see Expression Comparisons

Comparisons and Expression Primitives

string1 = string2	compare two strings for equality
string1 != string2	compare two strings: not equal
[[· · ·]]	see Expression Comparisons

Expression Comparisons: Strings

string = pattern	true if string matches the regular expression

Comparisons and Expression Primitives

string1 = string2	compare two strings for equality
string1 != string2	compare two strings: not equal
[[· · ·]]	see Expression Comparisons

Expression Comparisons: Strings

string = pattern	true if string matches the regular expression
string != pattern	true if string does not match the regular expr

Comparisons and Expression Primitives

string1 = string2	compare two strings for equality
string1 != string2	compare two strings: not equal
[[· · ·]]	see Expression Comparisons

Expression Comparisons: Strings

string = pattern	true if string matches the regular expression
string != pattern	true if string does not match the regular expr
string1 < string2	string1 less than string2 (lexicographically)

Comparisons and Expression Primitives

string1 = string2	compare two strings for equality
string1 != string2	compare two strings: not equal
[[· · ·]]	see Expression Comparisons

Expression Comparisons: Strings

string = pattern	true if string matches the regular expression
string != pattern	true if string does not match the regular expr
string1 < string2	string1 less than string2 (lexicographically)
string1 > string2	string1 greater than string2 (lexicographically)

Expression Comparisons, con't.

file1 -nt file2	file1 newer than file2

Expression Comparisons, con’t.

file1 -nt file2	file1 newer than file2
file1 -ot file2	file1 older than file2

Expression Comparisons, con’t.

file1 -nt file2	file1 newer than file2
file1 -ot file2	file1 older than file2
file1 -ef file2	file1 an equivalent name for file2

Expression Comparisons, con’t.

file1 -nt file2	file1 newer than file2
file1 -ot file2	file1 older than file2
file1 -ef file2	file1 an equivalent name for file2
expr1 -eq expr2	expression1 equals expression2

Expression Comparisons, con’t.

file1 -nt file2	file1 newer than file2
file1 -ot file2	file1 older than file2
file1 -ef file2	file1 an equivalent name for file2
expr1 -eq expr2	expression1 equals expression2
expr1 -ne expr2	expression1 not equal to expression2

Expression Comparisons, con't.

file1 -nt file2	file1 newer than file2
file1 -ot file2	file1 older than file2
file1 -ef file2	file1 an equivalent name for file2
expr1 -eq expr2	expression1 equals expression2
expr1 -ne expr2	expression1 not equal to expression2
expr1 -gt expr2	expression1 greater than expression2

Expression Comparisons, con’t.

file1 -nt file2	file1 newer than file2
file1 -ot file2	file1 older than file2
file1 -ef file2	file1 an equivalent name for file2
expr1 -eq expr2	expression1 equals expression2
expr1 -ne expr2	expression1 not equal to expression2
expr1 -gt expr2	expression1 greater than expression2
expr1 -ge expr2	expression1 greater than or equal to expression2

Expression Comparisons, con’t.

file1 -nt file2	file1 newer than file2
file1 -ot file2	file1 older than file2
file1 -ef file2	file1 an equivalent name for file2
expr1 -eq expr2	expression1 equals expression2
expr1 -ne expr2	expression1 not equal to expression2
expr1 -gt expr2	expression1 greater than expression2
expr1 -ge expr2	expression1 greater than or equal to expression2
expr1 -lt expr2	expression1 less than expression2

Expression Comparisons, con't.

file1 -nt file2	file1 newer than file2
file1 -ot file2	file1 older than file2
file1 -ef file2	file1 an equivalent name for file2
expr1 -eq expr2	expression1 equals expression2
expr1 -ne expr2	expression1 not equal to expression2
expr1 -gt expr2	expression1 greater than expression2
expr1 -ge expr2	expression1 greater than or equal to expression2
expr1 -lt expr2	expression1 less than expression2
expr1 -le expr2	expression1 less than or equal to expression2

Arithmetic

Arithmetic evaluation is done inside $((\cdots))$.

(*expr*)	prioritization

Arithmetic

Arithmetic evaluation is done inside $((\cdots))$.

(*expr*)	prioritization
-*expr*	negation

Arithmetic

Arithmetic evaluation is done inside $((\cdots))$.

(*expr*)	prioritization
-*expr*	negation
expr+*expr*	add

Arithmetic

Arithmetic evaluation is done inside $((\cdots))$.

(*expr*)	prioritization
-*expr*	negation
expr+*expr*	add
expr-*expr*	subtract

Arithmetic

Arithmetic evaluation is done inside $((\cdots))$.

(*expr*)	prioritization
-*expr*	negation
expr+*expr*	add
expr-*expr*	subtract
*expr***expr*	multiply

Arithmetic

Arithmetic evaluation is done inside $((\cdots))$.

(*expr*)	prioritization
-*expr*	negation
expr+*expr*	add
expr-*expr*	subtract
*expr***expr*	multiply
expr/*expr*	divide

Arithmetic

Arithmetic evaluation is done inside $((\cdots))$.

(*expr*)	prioritization
-*expr*	negation
expr+*expr*	add
expr-*expr*	subtract
*expr***expr*	multiply
expr/*expr*	divide
expr%*expr*	modulus

Arithmetic, con’t.

!*expr*	logical not

Arithmetic, con't.

!*expr*	logical not
expr || *expr*	logical or

Arithmetic, con't.

!*expr*	logical not
expr||*expr*	logical or
expr&&*expr*	logical and

Arithmetic, con’t.

!*expr*	logical not
expr||*expr*	logical or
expr&&*expr*	logical and
~*expr*	bitwise not

Arithmetic, con't.

!*expr*	logical not
expr || *expr*	logical or
expr&&*expr*	logical and
~*expr*	bitwise not
expr | *expr*	bitwise or

Arithmetic, con't.

!*expr*	logical not
expr || *expr*	logical or
expr&&*expr*	logical and
~*expr*	bitwise not
expr | *expr*	bitwise or
expr&*expr*	bitwise and

Arithmetic, con't.

!*expr*	logical not
expr || *expr*	logical or
expr&&*expr*	logical and
expr	bitwise not
expr | *expr*	bitwise or
expr&*expr*	bitwise and
*expr*ˆ*expr*	bitwise exclusive or

Arithmetic, con’t.

!*expr*	logical not
expr | | *expr*	logical or
expr&&*expr*	logical and
expr	bitwise not
expr | *expr*	bitwise or
expr&*expr*	bitwise and
*expr*ˆ*expr*	bitwise exclusive or
expr«*expr*	bitwise left shift

Arithmetic, con’t.

!*expr*	logical not
expr || *expr*	logical or
expr&&*expr*	logical and
expr	bitwise not
expr | *expr*	bitwise or
expr&*expr*	bitwise and
expr ˆ *expr*	bitwise exclusive or
expr«*expr*	bitwise left shift
expr»*expr*	bitwise right shift

Arithmetic, con't.

expr>=*expr*	compare: greater than or equal to

Arithmetic, con't.

expr>=expr	compare: greater than or equal to
expr<=expr	compare: less than or equal to

Arithmetic, con't.

expr>=*expr*	compare: greater than or equal to
expr<=*expr*	compare: less than or equal to
expr<*expr*	compare: less than

Arithmetic, con’t.

expr>=*expr*	compare: greater than or equal to
expr<=*expr*	compare: less than or equal to
expr<*expr*	compare: less than
expr>*expr*	compare: greater than

Arithmetic, con't.

expr>=expr	compare: greater than or equal to
expr<=expr	compare: less than or equal to
expr<expr	compare: less than
expr>expr	compare: greater than
expr==expr	compare: is equal

Arithmetic, con't.

expr>=*expr*	compare: greater than or equal to
expr<=*expr*	compare: less than or equal to
expr<*expr*	compare: less than
expr>*expr*	compare: greater than
expr==*expr*	compare: is equal
expr!=*expr*	compare: not equal

Arithmetic, con't.

expr>=*expr*	compare: greater than or equal to
expr<=*expr*	compare: less than or equal to
expr<*expr*	compare: less than
expr>*expr*	compare: greater than
expr==*expr*	compare: is equal
expr!=*expr*	compare: not equal
ident (op)= *expr*	assignment
	where op: +-*/%
	ex. x+= 3

Conditional Expression Primitives

Also for use in `[[···]]` or with the `test` command:

-r *file*	true if file exists and is readable

Conditional Expression Primitives

Also for use in `[[···]]` or with the `test` command:

-r *file*	true if file exists and is readable
-w *file*	true if file exists and is writable

Conditional Expression Primitives

Also for use in `[[···]]` or with the `test` command:

-r *file*	true if file exists and is readable
-w *file*	true if file exists and is writable
-x *file*	true if file exists and is executable

Conditional Expression Primitives

Also for use in `[[···]]` or with the `test` command:

-r *file*	true if file exists and is readable
-w *file*	true if file exists and is writable
-x *file*	true if file exists and is executable
-f *file*	true if file exists and is regular

Conditional Expression Primitives

Also for use in `[[···]]` or with the `test` command:

-r *file*	true if file exists and is readable
-w *file*	true if file exists and is writable
-x *file*	true if file exists and is executable
-f *file*	true if file exists and is regular
-d *file*	true if directory exists

Conditional Expression Primitives

Also for use in `[[···]]` or with the `test` command:

-r *file*	true if file exists and is readable
-w *file*	true if file exists and is writable
-x *file*	true if file exists and is executable
-f *file*	true if file exists and is regular
-d *file*	true if directory exists
-c *file*	true if file is character special

Conditional Expression Primitives

Also for use in `[[ ··· ]]` or with the `test` command:

-r *file*	true if file exists and is readable
-w *file*	true if file exists and is writable
-x *file*	true if file exists and is executable
-f *file*	true if file exists and is regular
-d *file*	true if directory exists
-c *file*	true if file is character special
-z *string*	true if length of string is zero

Conditional Expression Primitives

Also for use in `[[···]]` or with the `test` command:

-r *file*	true if file exists and is readable
-w *file*	true if file exists and is writable
-x *file*	true if file exists and is executable
-f *file*	true if file exists and is regular
-d *file*	true if directory exists
-c *file*	true if file is character special
-z *string*	true if length of string is zero
-n *string*	true if length of string is non-zero

Conditional Commands : IF

```
 if compound-list
 then
[ elif compound-list ]
[ elif compound-list ]
[ else compound-list ]
 fi
```

Example 28

```
if (( score < 65 )); then
  grade="F"
elif (( score < 80 )); then
  grade="C"
elif (( score < 90 )); then
  grade="B"
else
  grade="A"
fi
```

Conditional Commands : CASE

```
  case word in
  pattern [|pat[|pat ... ]]) compound-list ;;
[ pattern [|pat[|pat ... ]]) compound-list ;; ]
  esac
```

Example 29

```
case $TERM in
vt100)      PS1="Next: " ;;
iris-ansi) PS1="<esc>[33mNext<esc>[32m: ";;
*)          PS1="Hmmm...: ";;
esac
```

While Loops

```
while compound-list
do
  compound-list
done
```

Example 30

```
while read -r line
do
print -r "$line"
done
```

Until Loops

```
until compound-list
do
  compound-list
done
```

Example 31

```
integer i
i=0
until i > 4
do
  echo "  i=$i"
  i=i+1
done
```

For Loops

```
for var in list
do
  compound-list
done
```

Example 32

```
for filename in *.c
do
  echo ${filename}
done
```

Shell Parameters

$\$1 \cdots \$9$	positional parameter (i^{th} command-line argument)
shift	moves $\$2 \longrightarrow \$1$, $\$3 \longrightarrow \$2$, etc.
`${parameter:-word}`	value of parameter if it exists, `word` otherwsie
`${parameter:=word}`	assign default value
`${param:?word}`	display word if param is null or unset
`${param#pattern}`	remove small left pattern
`${param##pattern}`	remove large left pattern
`${param%pattern}`	remove small right pattern
`${param%%pattern}`	remove large right pattern

(see bashpat)

Shell Parameters, con't.

`${#param}`	string length of param value
`$# ${#*} ${#@}`	qty of positional parameters
`$@`	`"$1" "$2"`· · · `"$n"`
`$*`	`"$1 $2`· · · `$n"`
`RANDOM=$$`	initialize pseudo-random number generator
`$RANDOM`	get a pseudo-random number

Arrays

`typeset -u arrayname`	make an array
`arrayname[int]=`···	assign to an array
`${arrayname[int]}`	access an array

Example 33 *Printing a random card from a deck of cards (see bash04c)*

```
typeset -i i=0
typeset card
RANDOM=$$
# initialize the card array
for suit in clubs diamonds hearts spades; do
  for n in ace 2 3 4 5 6 7 8 9 10 jack queen king ; do
  card[i]="$n of $suit"
  i=i+1
  done
done
# print a random card from the deck
echo ${card[RANDOM%52]}
```

SysAdmin Inquiries

dmesg	examine kernel messages
fdisk -l	list partition tables for specified devices
sfdisk -l	list partitions of a device
cdrecord -scanbus	print inquiry strings for all SCSI devices
/proc/scsi/scsi	a file containing attached devices
df -T	display file system types, space, etc
fsck -N	show what fsck wants to check and repair
last	show all login attempts
lastb	show all bad login attempts
free	show memory usage information
fuser	show user of named files, sockets, etc
du / -bh | less	show detailed info on memory use for each subdirectory
cat /etc/issue	check what distribution you're using

SysAdmin Inquiries

cat /proc/interrupts	list interrupts in use
cat /proc/version	linux version and other info
cat /proc/filesystems	show types of filesystems in use
cat /etc/printcap | less	show set up of printers
lsmod	list modules in kernel (or /sbin/lsmod)
less /var/log/dmesg	see what dmesg dumped after last system bootup
chage -l loginname	see password expiry info
quota	see disk quota
sysctl -a | less	display all configurable linux kernel parameters
runlevel	print previous and current runlevel
init [runlvl]	switch runlevel

Creating a Dual Boot System #1

I will assume that you’re starting with a Windows computer

Create Space As Administrator, click on

- Control Panel : System and Maintenance : Administrative Tools : (double click) Computer Management. Provide our administrator password if requested.
- In the Navigation pane, under Storage, click Disk Management
- Right click on the volume you want to shrink, click on Shrink Volume.
- Follow instructions on the screen, and free up 20G to 100G of space.

Creating a Dual Boot System #2

Pick the system Choose one: (I prefer Scientific Linux, CUA has chosen Ubuntu)

- **Scientific Linux 6.3, 64-bit** At http://ftp1.scientificlinux.org/linux/scientific/6.3/x86_64/iso/, download SL-63-x86_64-2012-08-02-Everything-DVD1.iso and SL-63-x86_64-2012-08-02-Everything-DVD1.iso.
- **Scientific Linux 6.3, 32-bit** At http://ftp1.scientificlinux.org/linux/scientific/6.3/i386/iso/, download SL-63-i386-2012-08-02-Everything-DVD1.iso and SL-63-i386-2012-08-02-Everything-DVD1.iso.
- **Ubuntu 12.10** At http://www.ubuntu.com/download/desktop, pick the 32-bit or 64-bit version and then Get Ubuntu 12.10.

Creating a Dual Boot System #3

Burn DVDs Follow the directions at http://windows.microsoft.com/en-US/windows7/Burn-a-CD-or-DVD-from-an-ISO-file to burn the DVDs using the ISOs you've selected.

Reboot Reboot your computer *after inserting a DVD* (#1 for Scientific Linux). If your computer does not reboot using the DVD, you'll need to fix tell your BIOS to check your DVD reader for bootable disks. http://www.hiren.info/pages/bios-boot-cdrom gives an example (BIOS's may vary).

Follow Directions You'll be asked what language do you want, what sort of keyboard you want, what user-id you want, etc. For Scientific Linux, I usually choose a custom install and select most everything (I don't want my computer to be a web or ftp server). Scientific Linux will ask if you want to install Linux on the whole system (***DON'T** - unless you don't want Windows anymore*), if you want to re-install Linux (*unlikely*), or to install Linux on free space (*yes!*). With SL, in about an hour, it will eject Disk#1 and ask for Disk#2. You don't need to watch it while it does this.

Reboot Your new Linux will come up by default. You can change this by editing /boot/grub/grub.conf and changing default=0 to select the o/s you want as default.

Dual Boot Mac+Linux System

- Please note: I haven’t tried the following myself
- Please check on
  http://lifehacker.com/5934942/how-to-dual-boot-linux-on-your-mac-and-take-back-your-powerhouse-apple-hardware
  This webpage purports to describe how to make a dual boot mac+linux system