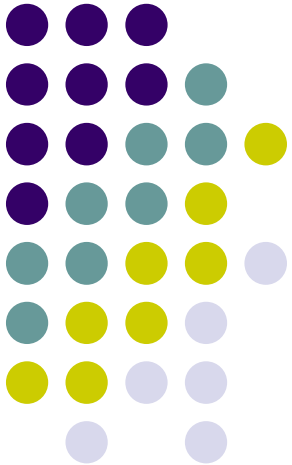
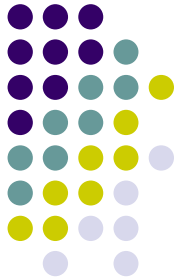
# Introduction to UNIX Frequently Used Commands

EE1356 Introduction to Information Systems

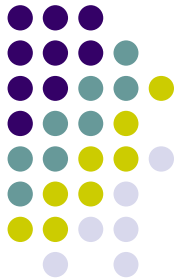


# Unix Commands

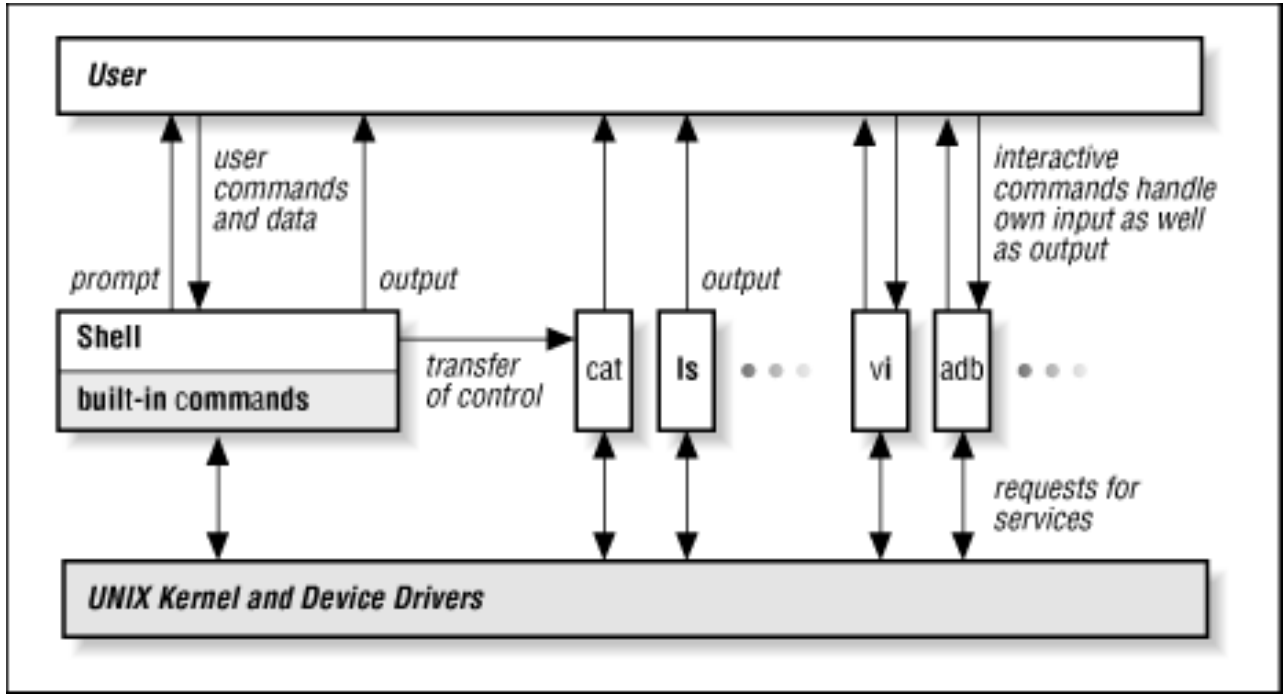


- Most Unix commands work on files and directories
- What you use might be a GNU Unix tools, especially in Linux
  - They are usually more powerful with more options
- The list of commands and options are definitely not complete.
- To become an advanced user, you need to
  - Use them frequently
    - install cygwin or Linux on your PC.
  - man
  - on-line reference and discussion
  - Books: UNIX Power Tools

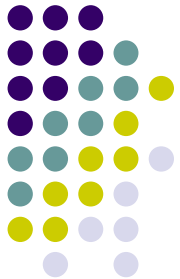
# Who Listens to What You Typed



- Shell --- command interpreter
  - Let users interact with UNIX.
  - For example, if you typed “ls *.txt”, the shell will find all *.txt files and pass them to ls.
  - Shell has many features for simplifying command line.

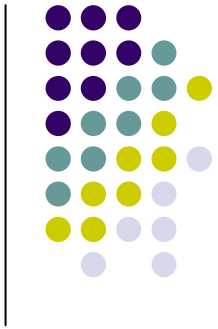


# Shell Defines the Working Environments



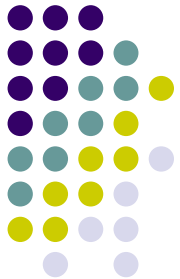
- Environment variables (shell variables)
  - Control the behavior of shell and programs
  - Expressed as *$VARIABLE_NAME*
  - For example, *$HOME* defines your home directories.
  - Can be setup at command line or the setup files for the shell (different shells have different setup files).

# There Are Many Shells



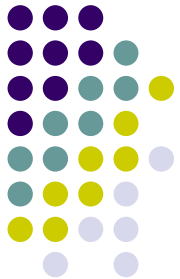
- Different flavor (features and working style) for different people
  - For interactive use, you will mostly choose bash or tcsh.
- sh
  - The Bourne shell (named after its creator, Steve Bourne).
  - The oldest: bit primitive and lacks job control features
  - Most UNIX users consider the sh superior for shell programming.
- csh
  - The C shell.
  - Developed for Berkeley UNIX.
  - A lot of nice features  (not in the sh), including job control and history

# There Are Many Shells



- ksh
  - The Korn shell (also named after its creator, David Korn).
  - Compatible with sh, but has most of the csh features plus some new features, like history editing.
  - Ksh is a standard part of UNIX System V Release 4 (SVR4).
- bash
  - The "Bourne-again" shell developed by the Free Software Foundation
  - bash is fairly similar to the Korn shell.
  - A standard on Linux.
- tcsh
  - Extended versions of the C shell.
  - Works like the original C shell - but with more features, and fewer mis-features.
- We will introduce other convenient shell features later.
- Now let’s first work on some commands.

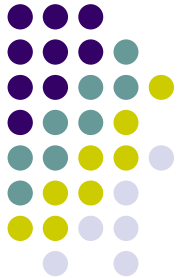
# pwd (Print Working Directory)



- Print the absolute pathname of the current working directory
- Example

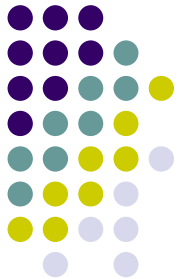
```
$ pwd
/home/queen
```

# cd: Change Directory



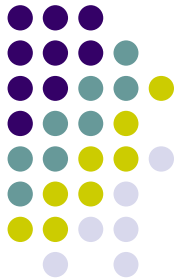
- "cd *dir*": Change the current directory to *dir*.
- The variable **HOME** is the default *dir*.
  - Note that $HOME has been setup automatically when you login the system
- The variable **CDPATH** defines the search path for the directory containing *dir*.
  - A convenient feature for switching directories.

# cd: Example



```
$ pwd
/home/peter
$ cd /usr/share/doc/HOWTO
$ pwd
/usr/share/doc/HOWTO
$ cd ../FAQ-Linux
$ pwd
/usr/share/doc/FAQ-Linux
$ cd ../../../lib
$ pwd
/usr/lib
$ cd ~queen
$ pwd
/home/queen
$ cd; pwd
/home/peter
```

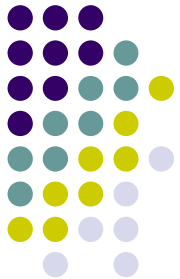
# $CDPATH: Example



- This example also shows how to set shell variables

```
$ export CDPATH="/usr:/usr/share:/home/peter"
$ cd lib;pwd
/usr/lib
$ cd local;pwd
/usr/local
$ cd lib;pwd
/usr/lib
$ export CDPATH=“./:/usr:/usr/share:/home/peter"
$ cd local;pwd
/usr/local
$ cd lib; pwd
/usr/local/lib
```

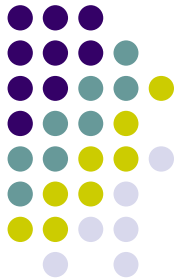
# echo and Shell Variables



- “echo”
  - to print strings on the standard output (i.e. screen)
  - likewise, to print shell variables (treat it like a string)

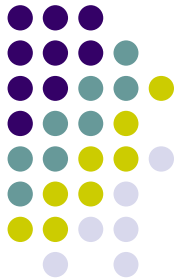
```
$ echo "This is a string"
This is a string
$ echo $HOME
/home/queen
$ echo $USERNAME
queen
$ echo Hello $USERNAME
Hello queen
```

# $PATH: A Very Important Shell Variable



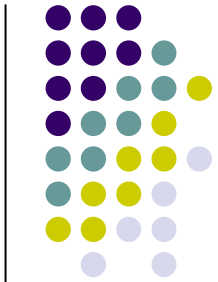
- $PATH sets the search paths of commands
  - Shell will search all and only those paths listed in the $PATH for a command
    - Commands are searched in the order listed; only the first one will be used.
    - Only these paths are searched; If commands are not in the $PATH, they will not be found
  - This is the single shell variables troubles most new users

# $HOME: Example and Setup Files



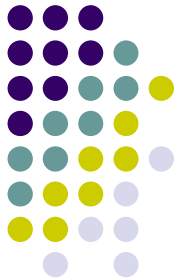
- bash
  - PATH="/usr/bin:/bin:/usr/local/bin:${PATH}"
- tcsh
  - setenv PATH="/usr/bin:/bin:/usr/local/bin:${PATH}"
- Most programs in UNIX has a setup file in $HOME
  - Filenames start with "."
  - For example, bash has a .bashrc and tcsh has a .tcshrc
  - Use "man" to find setup files and how to use them
  - You can copy other .*rc and edit for your own environments

# Shell Keyboard Shortcuts



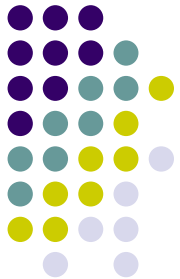
- Command history
  - View previous used commands with ↑ ↓ keys.
  - The ← and → arrow keys move the cursor left and right on the current line, allowing you to edit your commands.
  - Ctrl+A (Ctrl+E) bring you to the beginning (the end) of the current line.
  - The Backspace (Ctrl+H) and Del (Ctrl+D) keys work as expected.
  - Ctrl+K will delete from the position of the cursor to the end of line
  - Ctrl+W will delete the word before the cursor.
  - Ctrl+D on a blank line == exit.
  - Ctrl+C will interrupt the currently running command, or if you were editing your command line, it will cancel the editing and get you back to the prompt.
  - Ctrl+L clears the screen.
  - Ctrl+S and Ctrl+Q, which are used to suspend and restore output to the screen. They are not used very often, but you might suspend the current session by accidentally type Ctrl+S. Then try Ctrl+Q to return you the screen.

# Revisiting ls



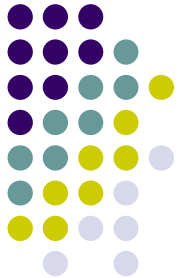
- -a: lists all files, including **hidden files** Remember that in UNIX , hidden files are those whose names begin with a .; the -A option lists “almost” all files, which means every file the -a option would print except for “.” and “..”
- -R: lists recursively, i.e. all files and subdirectories of directories entered on the command line.
- -s: prints the file size in kilobytes next to each file.
- -l: prints additional information about the files.
- -d: treats directories on the command line as if they were normal files rather than listing their contents.

# More File Handling Utilities



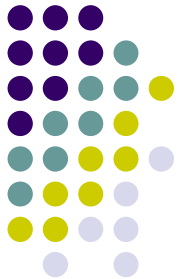
- mkdir: Creating Empty Directories
  - -p option.
    - Create parent directories if they did not exist before. Without this option, mkdir would just fail, complaining that the parent directories do not exist
    - Return silently if the directory which you want to create already exists. Similarly, without the -p option, mkdir will send back an error message, complaining that the directory already exists.
  - Example:
    - mkdir new_dir
    - mkdir –p new_parent/new_dir
- touch: Creating Empty Files
  - Example:
    - touch new_file

# More File Handling Utilities (Cont.)



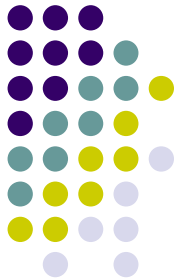
- rm: Deleting Files or Directories
  - -r, or -R: delete recursively. This option is **mandatory** for deleting a directory, empty or not. However, you can also use rmdir to delete empty directories.
  - -i: request confirmation before each deletion. rm is usually an **alias** to rm -i, for safety reasons.
  - -f, the opposite of -i, forces deletion of the files or directories, even if the user has no write access on the files.
  - Example
    - rm -i images/*.jpg
    - rm -Rf images/misc/ file*:
- mv: Moving or Renaming Files
  - -f: forces operation – no warning if an existing file is overwritten.
  - -i: the opposite. Asks the user for confirmation.
  - -v: **verbose** mode, report all changes and activity.
- cp: Copying Files and Directories
  - -R, -i, -f, -v

# Viewing File Contents



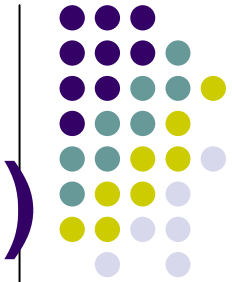
- cat
  - to print file contents on the standard outputs
- tail/head
  - print the last (first) 10 lines of a file to std outputs.
  - tail /var/log/mail/info
  - tail –n20  /var/log/mail/info (print the last 20 lines)
  - tail –f /var/log/messages (print to stdio whenever lines are added to the file)
- less is more
  - less and more to print file contents on stdio page by page (page size is defined by the current terminal)
  - a GNU program for traditional UNIX “more”;
  - It can go forward and backward.
  - less starts without reading the entire file; indispensible for viewing large files
  - Use “q” for quit and “h” for help.

# Changing File Attributes



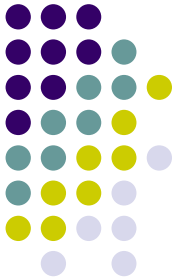
- chown, chgrp: Change the Owner and Group of One or More Files
  - -R: recursive.
  - -v: verbose mode. Displays all actions performed by chown whether changed or not.
  - -c: like -v, but only reports which files have been changed.

# Changing File Attributes (Cont.)

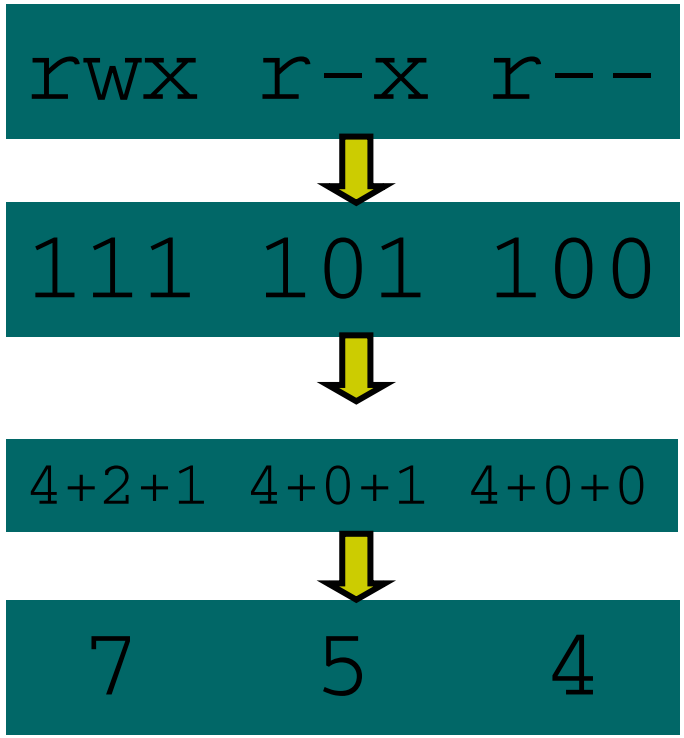


- chmod: Changing Permissions on Files and Directories
- Two forms of specifying options
  - in octal number
  - in character expression

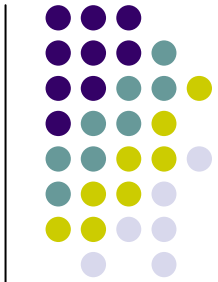
# chmod in octal



- The 9 permission characters in octal numbers
- Example:
- To change a file permission to rwxr-xr--, use “chmod 754 filename”

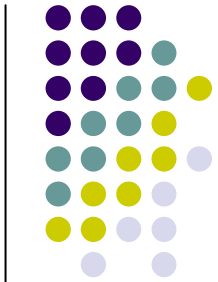


# chmod in Character Expression



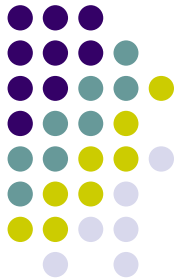
- The option format is in [category] <+|-|=> <permissions>
- [category]
  - u (User, permissions for owner)
  - g (Group, permissions for owner group)
  - o (Others, permissions for “others”)
  - a (all) or If no category is specified, changes will apply to all categories.
- <+|-|=>
  - + sets a permission,
  - - removes the permission
  - = sets the permission.
- <permissions>
  - r (Read)
  - w (Write)
  - x (eXecute)
- Example
  - chmod -R o-w /shared/docs:
  - chmod -R og-w,o-x private/:

# Shell Globbing Patterns



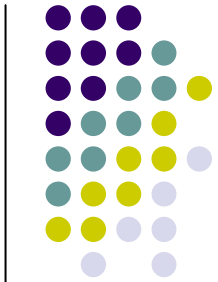
- File names in regular expression
  - A powerful way to indicate a group of files
  - ?: matches one and only one character
  - [...]: matches any character found in the brackets.
    - Characters can be referred to either as a range of characters (i.e. 1-9) or **discrete values**, or even both. Example: [a-zBE5-7] will match all characters between a and z, a B, an E, a 5, a 6 or a 7;
  - [!...]: matches any character **not** in the brackets.
    - [!a-z], for example, will match any character which is not a lowercase letter;
  - {c1,c2}: matches c1 or c2,
    - where c1 and c2 are also globbing patterns, which means you can write {[0-9]*,[acr]}.

# Shell Globbing Patterns Examples



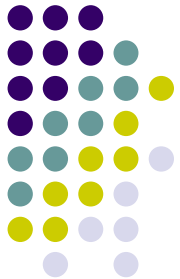
- /etc/*conf
  - all files in the /etc directory with names ending in conf.
  - It can match /etc/inetd.conf, /etc/conf.linuxconf, **and also** /etc/conf if such a file exists.
  - Remember that * can also match an empty string.
- image/{cars,space[0-9]}/*.jpg
  - all file names ending with .jpg in directories image/cars, image/space0, (...), image/space9, if those directories exist.
- /usr/share/doc/*/README
  - all files named README in all of /usr/share/doc's immediate subdirectories.
  - This will make /usr/share/doc/mandrake/README match, for example, but not /usr/share/doc/myprog/doc/README.
- *[!a-z]
  - all files with names that do **not** end with a lowercase letter in the current directory.

# Redirections and Pipes



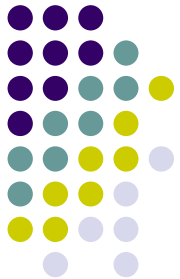
- Unix command usually handles three file descriptors :
  - standard input (keyboards), standard output (screen) and standard error (screen).
  - Their respective numbers are 0, 1 and 2.
- We can put these files to work together in creating more powerful tools
- Redirection (<>)
  - Redirect the input/output of commands to other files
  - For example, write the standard outputs to a real file
- Pipe (|)
  - A combination of input and output redirections.
  - For example, command A generates outputs to standard output (screen), which is redirected to standard inputs, and at the same time, command B takes inputs from standard input.
    - In the end, command B takes inputs from command A's outputs

# Redirection Examples



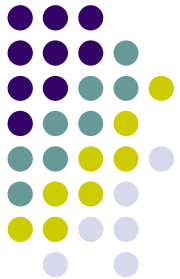
- ls images/*.png 1>file_list
- ls images/*.png >file_list
- wc -l 0<file_list
- wc -l <file_list
- sed -e 's/\.png$//g' <file_list >the_list
- ls -R /shared >/dev/null 2>errors

# Pipes Examples



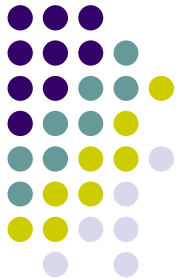
- ls images/*.png | wc -l
- ls images/*.png | sed -e 's/\.png$//g' >the_list
- ls images/*.png | sed -e 's/\.png$//g' | less
- xwd -root | convert - ~/my_desktop.png

# Shell Command Line Completion



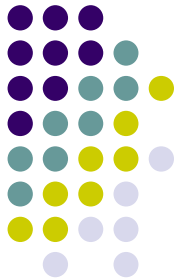
- Reduce the typed character for users
- Example:
  - Assume we have two files in the current directory: file_with_very_long_name_impossible_to_type and file_of_shorter_names
  - $ less fi<TAB>
  - $ less file_
  - less file_w<TAB>
  - less file_with_very_long_name_impossible_to_type

# More Shell Command Line Completion



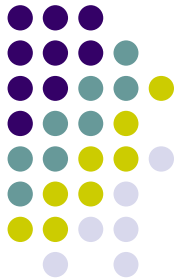
- Works in many places
- A <TAB> after an unfinished command will try to complete the command
- A <TAB> after the word preceded by a “magic” character
  - ~ complete a user name
  - @ complete a machine name
  - $ complete an environment variable
  - / complete a filename
  - ! complete a shell command

# More Shell Command Line Completion (Cont.)



- Esc-<x> (<x>= ~, @ , $, etc)
  - Attempt to come up with a unique completion. If it fails, it will complete the word with the largest possible substring in the choice list.
- Ctrl+x <x>
  - Displays the list of possible choices without attempting any completion.
- Example
  - To see all the environment variables defined: Ctrl+x $ on a blank line.
  - If you want to see the man page for the nslookup command,
    - you simply type man nsl then Esc-!, and the shell will automatically complete the command to man nslookup.

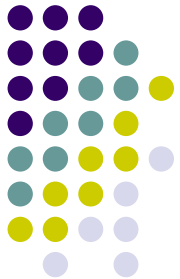
# Shell Job Control



- Foreground jobs
  - wait for the command to finish before the shell returns control to you.
- Background jobs
  - release the shell commands
  - use “Ctrl-Z” to put a foreground job to backgrounds
  - Add “&” to a to-be-sent-to-background command
- Example:

```
$ cp -R images/ /shared/ 2>/dev/null
  (Type C-z here)
  [1]+ Stopped cp -R images/ /shared/ 2>/dev/null
$ bg
  [1]+ cp -R images/ /shared/ 2>/dev/null &
$ cp -R images/ /shared/ 2>/dev/null &
$ fg 5 (or fg %5)
  (5 is job number, you can use “jobs” to check the job list of the current shell).
```

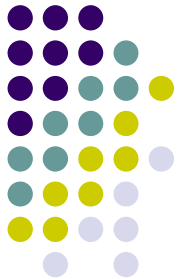
# Shell Alias



- Alias
  - Renaming of frequently used commands with options
  - Usually specified in .bashrc
- Examples

  alias rm='rm -i‘
  alias cp=‘cp -i'
  alias ls='ls -F --color=tty'
  alias dir='ls --color=auto --format=vertical'
  alias ll='ls -l'
  alias la='ls -A‘
  *Note that no “space” before or after “=“*
  *The quote is also important.*

# bash Shell Functions



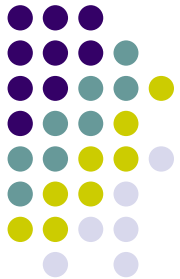
- Defining shell function is another way to create alias, but much more powerful
  - alias is simply string substitution
- Example
  - `la () { ls -a "$@"; }`

    *The spaces and the semicolon (;) are both important!*

    *The "$@" is replaced by the command-line arguments (other options, or directory and filenames)*

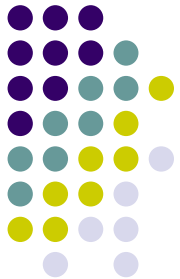
    If you already have an alias of “la”, you need to “unalias la” to create a function with a name of “la”

# Analyzing and Finding Files



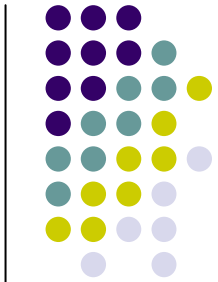
- grep: Locate Strings in Files
- fgrep: Locate Strings in Files with Patterns Listed in another file
- wc: Calculating Elements in Files
- sort: Sorting File Content
- find: Find Files According to Certain Criteria

# grep



- grep: Locate Strings in Files
  - “General Regular Expression Parser”
  - looks for a pattern given as an argument in one or more files.
  - -i: make a case insensitive search (i.e. ignore differences between lower and uppercase);
  - -v: invert search. display lines which do not match the pattern;
  - -n: display the line number for each line found;
  - -w: tells grep that the pattern should match a whole word.
  - Example:
    - grep postfix /var/log/mail/info
- There are many “grep” variations: egrep, fgrep, agrep, etc.

# fgrep



- fgrep: Locate Strings in Files with –f options

  # grep postfix /var/log/mail/info |grep status=sent

  *A clumsy way to find lines with both “postfix” and “status=sent” (meaning successfully sent mails)*

  # echo -e ’status=sent\npostfix’ >./patterns.txt

  *Create “patterns.txt” with two lines (“status=sent” and “postfix”)*

  # fgrep -f ./patterns.txt /var/log/mail/info

  *Use the strings in patterns.txt as the searching patterns*

  # echo ’peter@mandrakesoft.com’ >>./patterns.txt

  *Add another pattern*

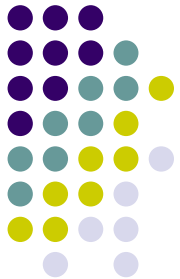
  # fgrep -f ./patterns.txt /var/log/mail/info

  *Search again*

  # fgrep -f ./patterns.txt /var/log/mail/info | tail -n2 | head -n1

  *Search again and show only the last one*

# wc

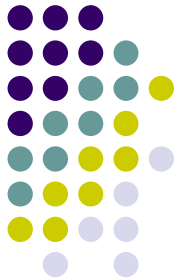


- wc: Calculating Elements in Files
  - to calculate the number of strings and words in files.
  - -l: print the number of new lines;
  - -w: print the number of words;
  - -m: print the character total;
  - -c: print the number of bytes;
  - -L: print the length of the longest line in the obtained text.
- Example

  # fgrep -f ./patterns.txt /var/log/mail/info | wc -l

  *Showing the number of lines with specified patterns (in the last example, the number of messages successfully sent to* peter@mandrakesoft.com

# sort



- sort: Sorting File Content
- Examples

  $ sort /etc/passwd

  *The sort command sorts data in ascending order starting by the first field (in our case, the login field) by default.*

  $ sort -r /etc/passwd

  $ sort /etc/passwd -t":" -k3 -n

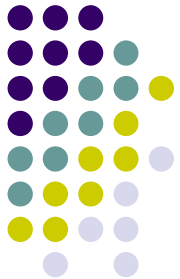
  *sort a file in ascending order using the UID field:*

  *-t":": the field separator is the ":" symbol*

  *-k3: sorting must be done by third column*

  *-n: the sort is to occur on numerical data, not alphabetical*

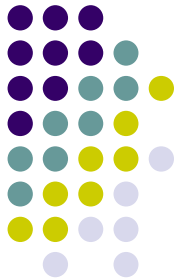
# find



- find: Find Files According to Certain Criteria
  - Scan one or more directories (and their sub-directories) and find files which match a certain criteria.
  - Syntax

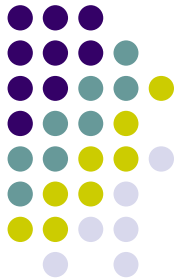
find [options] [directories] [criterion] [action]

# Problems of “find”ing



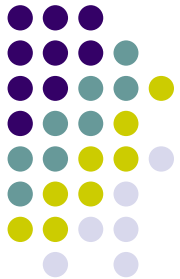
- Even though it is very useful, the syntax is truly obscure
  - What makes it worse is that UNIX vendors often changes options of “find”.
  - “-print” might not be there
    - For some “find”, nothing will show on the screen if “-print” is not there
    - Test your system’s find to see if “-print” is turned on by default
- Find will try to read every files’ inodes
  - It can be very slow if you are not careful
  - Use BSD find or GNU locate
    - But these depends on file database
- Learn with examples
  - Refer to Unix Power Tools

# find examples



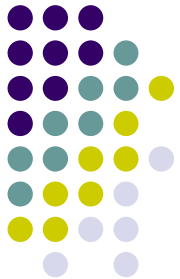
- **Looking for Files with Particular Names**
  - % **find . -name *.o -print**
  - % **find . -name '*.o' -print**
  - % **find . -name "[a-zA-Z]*.o" -print**
- A useful alias function
  - ff () { find . -name "$@" -ls; }

# find examples



- **Searching for Old Files**
  - % **find . -mtime 7 -print**
    - find a file that is seven days old
  - % **find . -type f -atime +30 -print**
    - list all files that have not been read in 30 days or more

# find examples



- **Use operators to combine search criteria**
  - **AND (-a), Or (-o), Not (!)**
  - **Use \( and \) to group criteria**
- **Examples**

  **% find . -name "*.o" -o -name "*.tmp" –print**

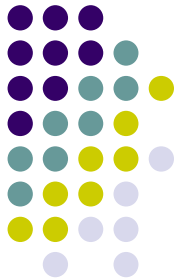
  **% find . -atime +5 -name "*.o"  -print**

  **% find . -atime +5 \( -name "*.o" -o -name "*.tmp" \) -print**

  **% find . ! \( -atime +5 \( -name "*.o" -o -name "*.tmp" \) \) -print**

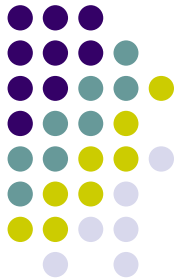
  *Note that all operators are separated with “spaces”, i.e. there are spaces before/after all operators*

# find times



- *The times that "find" finds are in days*
  - *A number with no sign*
    - *for example, 3 (as in -mtime 3 or -atime 3), means the 24-hour period that ended exactly three days ago*
    - *3 means between 96 and 72 hours ago.*
  - *A number with a minus sign (-) refers to the period since that time.*
    - *For example, -3 (as in -mtime -3) is any time between now and three days ago*
    - *-3 is between 0 and 72 hours ago.*
  - *A number with a plus sign (+) refers to the 24-hour period before that time.*
    - *For example, +3 (as in -mtime +3) is any time more than three days ago*
    - *+3 is more than 96 hours ago.*

# find exact times



- **Exact File Time Comparisons with -newer**
- Example

  % **touch 04251600 /tmp/4PMyesterday**

  % **find . -newer /tmp/4PMyesterday -print**

  *to find the files created after 4pm yesterday*

  % **touch 0703104682 /tmp/file1**

  % **touch 0804213785 /tmp/file2**

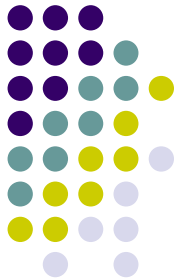
  *Check your touch manual for y2k bugs*

  % **find . -newer /tmp/file1 ! -newer /tmp/file2 -print**

  *to find the files created between file1 and file2*

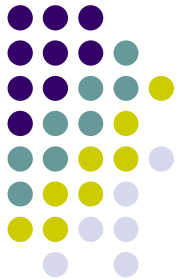
  % rm /tmp/file[12]

# Running Commands on What You Find



- % **find . -exec echo {} \;**
  - {} represents the files found
  - \; to end the command issued by –exec ( you can also use ";")
- % **find . -perm -20 -exec chmod g-w {} \;**
- % **find . -perm -20 -print | xargs chmod g-w**
  - *xargs* reads a group of arguments from its standard input, then runs a UNIX command with that group of arguments.
  - It keeps reading arguments and running the command until it runs out of arguments

# Finding Many Things with One Command

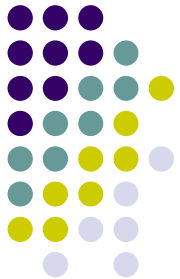


- A find command with multiple -exec

```
$ find . \( -type d -a -exec chmod 771 {} \; \) -o \
  \( -name "*.BAK" -a -exec chmod 600 {} \; \) -o \
  \( -name "*.sh" -a -exec chmod 755 {} \; \) -o \
  \( -name "*.txt" -a -exec chmod 644 {} \; \)
```

- Note that find arguments are just evaluation operations
  - They can be combined arbitrarily

# Finding files with grep



- % **find . -type f -print | xargs fgrep '$12.99' /dev/null**
  - The last /dev/null is a empty device (file). It can also take anything and save nothing (still empty).
  - grep does not print the filename if there is only one file to be processed
  - xargs runs the command as the follows

    fgrep '$12.99' /dev/null ./afile ./bfile ...

    fgrep '$12.99'
    /dev/null ./archives/ad.0190 ./archives/ad.0290 ...

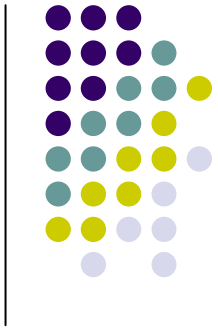
    fgrep '$12.99' /dev/null ./old_sales/ad.1289
  - Output will be like this

    ./old_sales/ad.0489: Get it for only $12.99!

    ./old_sales/ad.0589: Last chance at $12.99, this month!

    ./old_sales/ad.1289: Get it for only $12.99 today.

# find options

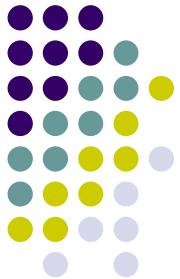


- find [options]
  - -xdev: do not search on directories located on other file systems
  - -mindepth <n>: descend at least <n> levels below the specified directory before searching for files
  - -maxdepth <n>: search for files which are located at most n levels below the specified directory
  - -follow: follow symbolic links if they link to directories. By default, find does not follow them
  - -daystart: when using tests related to time, take the beginning of current day as a timestamp instead of the default (24 hours before current time).

# find search criteria

- find [criterion]
  - -type <type>: search for a given file type. <type> can be one of: f (regular file), d (directory), etc.
  - -name <pattern>: find files whose names match the given <pattern>. (a **shell globbing** pattern)
  - -iname <pattern>: like -name, but ignore case
  - -atime <n>, -amin <n>: find files that have last been accessed <n> days ago (-atime) or <n> minutes ago (-amin). You can also specify +<n> or -<n>, in which case the search will be done for files accessed at most or at least <n> days/minutes ago;
  - -anewer <file>: find files which have been accessed more recently than file <file>
  - -ctime <n>, -cmin <n>, -cnewer <file>: same as for -atime, -amin and -anewer, but applies to the last time that the contents of the file were modified
  - -regex <pattern>: simliar to -name, but work on a complete path
  - -iregex <pattern>: same as -regex, but ignore case.

# find search criteria rules

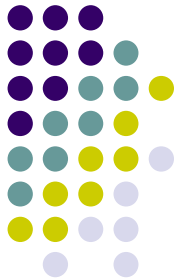


- find [criterion] combination rule
  - <c1> -a <c2>: true if both <c1> and <c2> are true
    - -a is implicit, therefore you can type <c1> <c2> <c3> ... if you want all tests <c1>, <c2>, ... to match
  - <c1> -o <c2>: true if either <c1> or <c2> are true, or both.
    - Note that -o has a lower **precedence** than -a, therefore if you want to match files which match criteria <c1> or <c2> and match criterion <c3>, you will have to use parentheses and write ( <c1> -o <c2> ) -a <c3>. You must **escape** (deactivate) parentheses \) and \(, as otherwise they will be interpreted by the shell !
  - -not <c1>: inverts test <c1>, therefore -not <c1> is true if <c1> is false.

# find actions

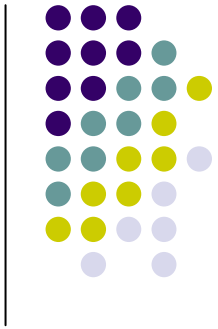
- find [action]
  - -print: just prints the name of each file on standard output. This is the default action.
  - -ls: prints on the standard output the equivalent of ls -ilds for each file found.
  - -exec <command>: execute command <command> on each file found. The command line <command> must end with a ;, which you must escape so that the shell does not interpret it; the file position is marked with {}. See the usage examples.
  - -ok <command>: same as -exec but ask confirmation for each command.

# File Archiving and Data Compression



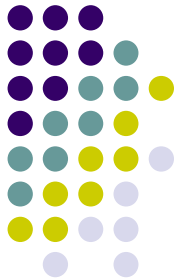
- tar: Tape ARchiver
  - Collect all files/directories under a directory into a single file (so can be put on a tape for backup)
- tar has a strange syntax, we will also learn it by examples

# tar examples



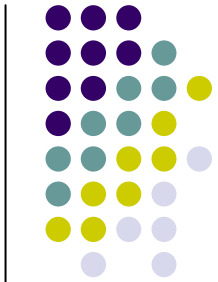
- $ tar cjf ~/images.tar.bz2 images/
  - *to create  (c option) the "~/images.tar.bz2" file (follow by f) with compression (j option) from the "images/" directories.*
- $ tar tjvf images.tar.bz2
  - *to test (t option) and print (v option) the files/directories in the "images.tar.bz2" file (follow by f) with decompression (j option).*
- $ tar xjvf images.tar.bz2
  - *to extract (x option) and print (v option) the files/directories in the "images.tar.bz2" file (follow by f) with decompression (j option).*
  - *Note that this will overwrite files/directories in the current dir.*

# tar options



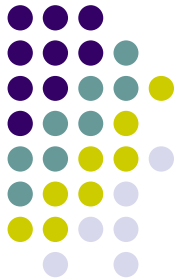
- c: this option is used in order to create new archives
- x: this option is used in order to extract files from an existing archive
- t: list files from an existing archive
- v: list the files which are added to an archive or extracted from an archive, or, in conjunction with the t option (see above), it outputs a long listing of files instead of a short one
- f <file>: create archive with name <file>, extract from archive <file> or list files from archive <file>.
  - If the file parameter is - (a dash), the input or output (depending on whether you create an archive or extract from one) will be associated to the standard input or standard output
- z: tells tar that the archive to create should be compressed with gzip, or that the archive to extract from is compressed with gzip
- j: same as z, but the program used for compression is bzip2
- p: when extracting files from an archive, preserve all file attributes, including ownership, last access time and so on. Very useful for file system dumps.

# Data Compression and Decompression Programs



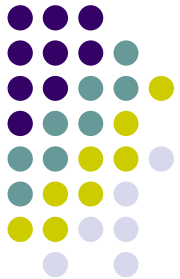
- gzip and bzip2
- options
  - -1, ..., -9: set the compression ratio. The higher the number, the better the compression, but better also means slower.
  - -d: uncompress file(s). This is equivalent to using gunzip or bunzip2.
  - -c: dump the result of compression/decompression of files given as parameters to the standard output.
- Note that By default, both gzip and bzip2 erase the file(s) that they have compressed (or uncompressed) if you don't use the -c option.
  - bzip2 has a –k option to avoid this
- Examples
  - $ bzip2 -9 afile.txt
    - At the output, there will be afile.txt.gz file (no more afile.txt).
  - bzip2 -dc images.tar.bz2 | gzip -9 >images.tar.gz
    - Recompression using gzip format

# Processes and Signals



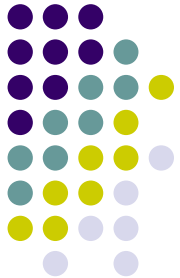
- All processes that run on a UNIX system are identified by a integer number (Process ID or PID).
  - PIDs are not associated with a shell (like job number)
- Signal is a way to inform processes of their status (for example Ctrl-C).
  - We can use names or integers to identify signals
    - For example SIGKILL or 9 can be used to kill a process

# Listing Processes



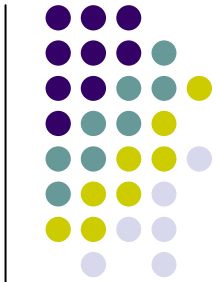
- ps
  - BSD and SVR4 has totally different options
- Fortunately you don't care too much about many other options
  - ps aux | less
  - ps –ef | less
- ps options
  - a: also displays processes started by other users;
  - x: also displays processes with no control terminal or with a control terminal different to the one you are using;
  - u: displays for each process the name of the user who started it and the time at which it was started.

# Sending Signals to Processes



- kill
- Example
  - $ kill -9 785
  - Send a KILL signal to process 785

# Viewing Processes

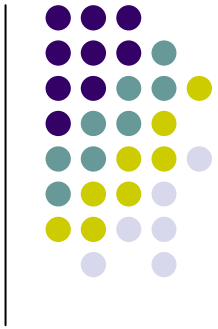


- top

peter@dhcp100.mandrakesoft.com: /home/peter

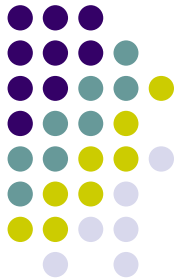
```
 12:48pm  up  8:05,  2 users,  load average: 1.40, 1.40, 1.31
97 processes: 93 sleeping, 4 running, 0 zombie, 0 stopped
CPU states:  1.9% user,  1.9% system, 96.0% nice,  0.0% idle
Mem:   192072K av,  181432K used,   10640K free,       0K shrd,    5124K buff
Swap:  249472K av,   52872K used,  196600K free                   55104K cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME COMMAND
 1618 fabman    16   1 14140  13M   296 R N  96.9  7.0 436:41 setiathome
20340 root      9   0 21400  12M  2428 S     1.1  6.7   0:04 X
20576 peter     9   0 11660  11M 10620 S     0.5  6.0   0:00 kdeinit
20632 peter    12   0  1052 1052   820 R     0.5  0.5   0:00 top
20633 peter     9   0 13336  13M 11196 S     0.3  6.9   0:01 ksnapshot
20579 peter     9   0 16716  16M 14584 S     0.1  8.7   0:01 kdeinit
    1 root      8   0   124   76    64 S     0.0  0.0   0:03 init
    2 root      9   0     0    0     0 SW    0.0  0.0   0:01 keventd
    3 root      9   0     0    0     0 SW    0.0  0.0   0:00 kapmd
    4 root     19  19     0    0     0 SWN   0.0  0.0   0:00 ksoftirqd_CPU0
    5 root      9   0     0    0     0 SW    0.0  0.0   0:03 kswapd
    6 root      9   0     0    0     0 SW    0.0  0.0   0:00 bdflush
    7 root      9   0     0    0     0 SW    0.0  0.0   0:00 kupdated
    8 root     -1 -20     0    0     0 SW<   0.0  0.0   0:00 mdrecoveryd
   12 root      9   0     0    0     0 SW    0.0  0.0   0:00 kjournald
   57 root      9   0   584  488   408 S     0.0  0.2   0:01 devfsd
  200 root      9   0     0    0     0 SW    0.0  0.0   0:00 kjournald
```

# top options



- k: to send a signal to a process. top will then ask you for the process' PID followed by the number of the signal to be sent (TERM— or 15— by default)
- M: to sort by the amount of memory they take up (%MEM)
- P: to sort by the CPU time they take up (%CPU; default sort method)
- u: this one is used to display a given user's processes
- i: by default, all processes, even sleeping ones, are displayed. This command ensures that only processes currently running are displayed (processes whose STAT field states R, Running) and not the others. Using this command again takes you back to showing all processes.
- r: to change the priority of selected process.

# Setting Priority to Processes



- nice, renice
- Every process in the system is running with defined priorities (also called “nice value”).
  - This value may vary from -20 to +20.
  - The maximum priority value is -20.
  - The more towards -20 → higher priority
  - Processes with any negative values use more system resources than others.
  - Processes with minimal priority (+20) will work when the system is not used by other tasks.
- Normal users may only lower the priority of processes they own within a range of 0 to 20.
- The super-user (root) may set the priority of any process to any value.
- Examples:
  - nice -n 19 dd if=/dev/cdrom of=~/mdk1.iso
  - renice +15 785
  - renice +20 -u peter