

**A WMP GmbH Training**  
presented in cooperation with  
**Allasso GmbH**

**Unix tools and software compilation**

---

**written by**  
**Klaus Weidner**  
**(kw@w-m-p.com)**

# Unix: First Contact

---

- **Unix is fundamentally a multiuser and multitasking operating system.**
  - **A login process is needed to authenticate the user**
  - **Several interactive users can be active at the same time**
  - **The *root* user has all privileges - including the ability to totally screw up the system.**
  - **Ordinary users have limited privileges - usually, they cannot manipulate files or processes belonging to other users.**
- **"Everything is a file"**
  - **Devices are represented by "special files", usually in /dev**
  - **Some Unix systems (i.e. Solaris and Linux) also have a /proc filesystem displaying process status information as (virtual) files**

# File system navigation

---

- The essential commands for navigating the Unix file system:
  - `cd` changes the current directory
    - specify absolute (`cd /etc`) or relative (`cd ../tmp`) paths
    - `cd ..` moves up a level
    - `cd` without arguments returns to the home directory
  - `pwd` prints the current working directory
  - `ls` lists the files in a directory
- Unix filesystems differ from most other operating systems:
  - Path components are separated with a '/' (forward slash).
  - File and directory names are cAsE SenSitIvE; *file*, *File* and *FILE* can coexist in the same directory.
  - There are no drive letters or other filesystem prefixes - filesystems are added to the root filesystem by *mounting* them as a subdirectory.

# ls examples

---

- **Show long output in single pages:**

```
# ls -l | more
```

- **sort output by time (-t), newest files last (-r):**

```
# ls -ltr
```

- **find newest file:**

```
# ls -tr | tail -1
```

- **sort by size (fifth column, numeric):**

```
Solaris: # ls -l | sort -k 5n
```

```
Linux (GNU ls): # ls -lSr
```

# Using `ls`

---

- By default, `ls` shows the filenames only. Use `-l` to view details.
- The first column shows the type and permissions:
  - The first letter is the type ('-': file, 'd': directory, 'l': symlink; other characters for special files such as devices)
  - The next three blocks of three characters show the access permissions.
- The link count is  $>1$  for hard links (discussed later).
- The owner and group are set when the file is created, and are not updated if someone else writes to it.

<i>type/</i>	<i>lnk</i>								
<i>permission</i>	<i>cnt</i>	<i>userid</i>	<i>group</i>	<i>size</i>	<i>date</i>			<i>filename</i>	
drwxrwxr-x	2	root	sys	1536	Jun 11 23:51			init.d	
-rw-r--r--	1	root	sys	664	Jun 11 22:31			passwd	
-r-----	1	root	sys	246	May 23 18:43			shadow	
lrwxrwxrwx	1	root	root	15	May 23 17:03			wtmp -> ../var/adm/wtmp	

# Lab Exercise: Unix navigation

1. Log in
2. Use *cd*, *ls* and *pwd* to look through the file system
3. Change to the */etc* directory
4. Show the directory contents
5. Sort the files by date
6. Which is the largest file?
7. (*Advanced*) Which files in */etc* have been modified since the operating system installation?

# *man* - the Unix online manual pages

---

- The `man` command displays the online manual pages.
  - Man pages are intended as references - they are not very useful as tutorials, but helpful if you need to look up command line switches.
  - Examples:

```
# man ls
# man man
```
- Useful command line options:
  - `-a` : show all matching manual pages
  - `-k` : search for keyword in the command descriptions.

Example:

```
# man -k directory
```

# ***man*: Solaris issues**

---

- **A standard Solaris installation doesn't include manual pages - they are in the *SUNWman* package.**
- **If 'man -k' doesn't work ("windex: No such file or directory"), run the following command to build the index (it takes a few minutes):**  
**# catman**

# Emergency *vi* survival guide

---

- The *vi* editor distinguishes between three operating modes - you cannot always use editing keys or simply start typing.
  - *command* mode: move cursor, edit existing text
  - *input* mode: enter new text
  - *extended* mode: commands that affect the entire file
- Start the editor with `vi FILE`. It is initially in command mode.
  - Arrow keys (or `hjkl`) move the cursor, `x` and `X` delete characters, `dd` deletes lines.
- To enter new text, type `i` to switch to input mode.
  - Cursor keys don't work - use `<Backspace>` only.
  - Use the `<Esc>` key to return to command mode.
- Extended commands start with a colon (`:`) and are confirmed with the `<Return>` key. `<Backspace>` edits the command.
  - `:wq` : save the file and exit
  - `:q!` : exit without saving (discard changes)

# More *vi* commands

---

- **There are several keys that switch from command to input mode:**
  - **i (a) : add text before (after) cursor position**
  - **I (A) : add text at start (end) of line**
  - **o (O) : open new line below (above) current line**
  - **C : delete from cursor to end of line, replace with new text**
- **Moving through the file in command mode:**
  - **/ (? ) : search forward (back) for text**
  - **n : repeat last search**
  - **Ctrl-~~f~~ (Ctrl-b) - move one screen forward (back)**
- **Other command mode keys:**
  - **. : repeat last editing command (i.e. dd...)**
  - **u : undo editing command (once only with standard VI)**
  - **U : restore current line**
  - **J : join current line and next line**
  - **7>> (7<<) : indent (unindent) seven lines**

# *vi* cut&paste, and extended commands

---

- **Cutting and pasting text in command mode**
  - **`dd` (`yy`)** : delete (copy) current line
  - **`5dd` (`5yy`)** : delete (copy) five lines (current + next four)
  - **`p` (`P`)** : paste deleted or copied lines below (above) cursor
- **Useful extended commands:**
  - **`:42`** : move to line 42
  - **`:$`** : move to end of file
  - **`:%s/^/#/` (`:%s/^#//`)** : insert (delete) '#' at start of line
  - **`:%s/cat/dog/g`** : globally replace *cat* with *dog*
  - **`:1,.d`** : Delete everything from the first line to the current line
  - **`:.,$d`** : Delete from the current line to end of the file
  - **`:%g/Error:/d`** : Delete all lines containing the string 'Error:'
  - **`:w newname`** : save copy of file, keep editing original
  - **`:e!`** : Discard changes and reload original file
  - **`:wq!`** : force overwriting a read-only file

# Lab Exercise: *vi*

1. Use `cd /var/tmp` to move to a temporary directory.
2. Create a new file using `vi test.txt`
3. Type *i* to switch to input mode, and enter at least three lines of text (i.e. your name and address). Don't try to fix any mistakes yet.
4. Practice switching between input and command mode. Use `x` to delete mistyped characters, and switch back to input mode to enter the correct text.
5. Try searching for a word.
6. (*Advanced*) Use the cut+paste functions to move lines around.
7. (*Advanced*) Replace all "e"s with "a"s.
8. Save the file and exit.

# Unix permissions

---

- The access permissions for each file or directory are determined by a bitmap. There are separate Read, Write, and eXecute (rwx) permissions.
  - The permissions don't apply to *root*, who always has access.
- The access permitted when a bit is set is different for files and directories:

	File	Directory
<i>Read</i>	Read the file contents	Read the list of files in the directory, i.e. when running <i>ls</i>
<i>Write</i>	Modify the file contents	Create new files or delete existing files. You don't need write access to a file to delete it.
<i>eXecute</i>	Run the file or script as a program	Use the directory as a component in a pathname, i.e. <i>/tmp/unreadabledir/file</i>

# Permissions and *chmod*

---

- The "rwx" permissions are maintained separately for the file owner, group and for others:
  - If you are the file owner, the *owner* permissions apply.
  - If you are not the owner, but are a member of a group corresponding to the file's group ID, the *group* permissions apply.
  - Otherwise, the *other* permissions apply.
- The *first* set of applicable permissions is used, not necessarily the most permissive one.
  - Used on FTP servers where anonymous uploads are readable by everyone *except* the anonymous FTP user(s).
- The *chmod* command is used to change access rights:

```
# chmod +x script.sh
# chmod 4750 tape_manager
# chmod u+s,o-rwx tape_manager
```

# An additional set of permission bits is used for special purposes

---

- The *sticky bit* should be set on public directories (i.e. */tmp*) to restrict file deletion to the file's owner.
- If the *setuid* and/or *setgid* bits are set on an executable file, it is run with the file's owner and/or group ID instead of the caller's privileges. This is used to give additional rights to users for a specific purpose, and is a very frequent source of security problems.
- Setting the *setgid* bit on a directory causes all files and directories created in it to inherit the group permissions, which is very useful for files shared by a group of users.
- Setting the *setgid* bit on a non-executable file marks a file for POSIX mandatory locking, which can cause programs accessing it to unexpectedly block.

# Unix permission summary table

---

Octal code	Permissions	<i>chmod</i> flag	<i>ls</i> display
0001	other eXecute	<b>o+x</b>	-----x
0002	other Write	<b>o+w</b>	-----w-
0004	other Read	<b>o+r</b>	-----r--
0010	group eXecute	<b>g+x</b>	-----x---
0020	group Write	<b>g+w</b>	-----w----
0040	group Read	<b>g+r</b>	-----r-----
0100	owner eXecute	<b>u+x</b>	---x-----
0200	owner Write	<b>u+w</b>	--w-----
0400	owner Read	<b>u+r</b>	-r-----
1000	sticky bit	<b>+t</b>	-----t (other exec) -----T (no other exec)
2000	set group ID	<b>g+s</b>	-----s---- (group exec) -----S---- (no group exec)
4000	set user ID	<b>u+s</b>	---s----- (owner exec) ---S----- (no owner exec)

# The *umask* modifies the permissions of new files

---

- Each process has a *umask* setting. It is a bitmask (usually shown as an octal number) that *removes* permission bits from any files or directories that the process creates.
  - The umask "022" removes write access for *group* and *other*.
  - If each user has a private group, the umask "002" can be used; this makes it easier for groups to share access to directories.
  - The umasks "077" removes *all* rights for group and other, and can cause problems if used indiscriminately.
- The `umask` command is usually set in `/etc/profile`, and can be overridden in the user's personal `.profile`.
- Each process inherits the parent's umask. If a process has an unexpected umask setting, this can be caused by:
  - a configuration setting, i.e. `/etc/default/ftpd`
  - the parent's settings, i.e. an entry in the `/etc/init.d/` startup file.

# Changing users and groups with *chown* and *chgrp*

---

- **chown** changes the owner of a file or directory, and can be used only by the *root* user (on most systems). Example:  

```
# chown root uploaded_file
```
- **chgrp** changes the file's group. Non-*root* users can only reassign files to groups of which they are members.
- Most systems support setting the user and group at the same time:
  - **Solaris:** `chown user:group FILE`
  - **Linux:** `chown user.group FILE`
- The **-R** flag applies the setting to a directory recursively.  
Example:  

```
# chgrp -R firewall /etc/fw
```

# Symlinks are pointers to filesystem objects

---

- **Symlinks (soft links) are pointers to path names that are dereferenced when accessing a file.**
  - **Symlinks can refer to files, directories or anything else in the filesystem**
  - **`ls -l` displays symlink targets with a "`->`" symbol.**
  - **The owner and permissions of the symlink are ignored - the rights for the link *target* determine what access is possible.**
  - **Deleting or renaming the file pointed to results in an unusable "dangling symlink".**
- **Symlinks are created with the command `ln -s TARGET LINK`.**
  - **The `ln -s` command does *not* verify if the target is valid.**
  - **To create a link to a file in a different directory, `cd` to the link target directory, and use a relative path:**

```
# cd /etc
# ln -s ../opt/CPFW1-41 fw
```

# Hard links allow multiple names for files

---

- **Hard links create multiple fully equivalent ways to refer to the same file - the original and the link are indistinguishable.**
  - **`ln FILE NEWNAME` creates a hard link.**
  - **The hard links can be deleted in any order, and the file data remains until all links are gone.**
  - **No one (not even *root*) is allowed to hardlink directories.**
  - **Hard links can only be created within a single file system.**
- **Unix filesystems store data in *inodes*, and directories store (name, inode number) pairs. A *hard link* occurs when multiple directory entries refer to the same inode number.**
  - **`ls -li` displays inode numbers - useful to verify which files were linked.**
- **The metadata (owner, permissions, datestamps etc.) are stored in the inode, and therefore the same for all links.**

# The Unix shell(s)

---

- **The command shell is the most important Unix user interface.**
  - **A powerful shell makes system administration more efficient and more reliable.**
  - **Unfortunately, commercial Unix systems ship with ancient shells.**
  - **Installation of a modern open-source shell (*bash* or *zsh*) is highly recommended.**
- **Shell scripts are used for many automation tasks.**
  - **The bootup process is controlled by shell scripts in `/etc/init.d/`**
  - **Creating small scripts helps make repetitive tasks more efficient and less error-prone.**

# Using *zsh*

---

- ***zsh* offers many useful interactive features:**
  - **Cursor keys work by default for command line editing**
  - **Filename completion using the <Tab> key**
- **Useful keys:**
  - **<Esc>-. : retrieve last word of previous line (repeat for older lines)**
  - **<Esc>-Q : save current command line, prompt for new command**
  - **<Ctrl>-R : search previous command lines**

# Recommended *zsh* configuration

---

```
# set prompt: hostname, last 20 chars of path, # (for root) or %
PS1='%M:%20<\<\<<%~%# '

# aliases and functions
alias ll='ls -lFA'
alias l='ls -CF'
nd () { mkdir -p "$1"; cd "$1"; }

# useful options
setopt auto_list no_auto_menu
setopt no_bang_hist no_beep rm_star_silent

# directory stack - usage: cd /tmp; dh; ls ~1; popd
setopt auto_pushd pushd_silent pushd_to_home
alias dh="dirs -v"
```

# Life with *ksh*

---

- *ksh* is arguably the least brain-dead shell shipped with Solaris; unfortunately the default configuration is not helpful.
- *ksh* supports two editing modes: *vi*- and *emacs*-compatible.
- *vi* mode is activated with `set -o vi`, it is stateful like the VI editor.
  - cursor keys don't work, use `hjkl`.
  - Use the `'/'` key to search previously entered command lines.
- *emacs* mode is activated with `set -o emacs`, it is fairly intuitive even without Emacs skills.
  - Default configuration uses `<Ctrl>-fbnp` for cursor motion; see next slide about activating cursor keys.
  - Use `<Esc><Esc>` for filename completion.
  - `<Esc>=` lists matching names.
  - Use `<Ctrl>-R` to search previous command lines.

# Recommended *ksh* configuration

---

- The following configuration activates command line editing using the cursor keys; a per-window history file; and some useful aliases.

## **\$HOME/.profile**

```
PATH=$PATH:$HOME/bin
export PATH

TTY=`tty`
HISTFILE=$HOME/.sh_history.`basename $TTY`
HISTSIZE=100
export HISTFILE HISTSIZE

if [ -z "$ENV" ]
then
  ENV=$HOME/.kshrc
  export ENV
  exec ksh
fi
```

## **\$HOME/.kshrc**

```
set -o emacs
alias __A='echo "\020"'
alias __B='echo "\016"'
alias __C='echo "\006"'
alias __D='echo "\002"'
HOSTNAME=`hostname`
PS1='$HOSTNAME:$PWD\# '
alias ll='ls -lFA'
alias l='ls -CF'
alias m='more'
```

# Unix Filenames

---

- **Unix Filenames can contain any ASCII characters except '/' and ASCII NUL.**
  - **The kernel normally doesn't care about codepages or character sets - the filename is just a byte stream.**
  - **Applications are free to interpret the bytes as ISO-8859-1 (Latin-1), Unicode UTF8 encoding or anything else.**
- **Attackers frequently use directories with unusual filenames to hide data.**
  - **A typical example is '. . . ', which isn't immediately noticeable in directory listings.**
- **Filenames containing spaces or newlines can cause reliability and security problems, especially if they are used incautiously in shell scripts.**

# Shell Quoting

---

- **If special characters (spaces, shell metacharacters such as ( ) " ' \* ? [ ] < > or unprintable control characters) appear in filenames, be careful when manipulating them with the shell.**
  - **Shell quoting using " or ' follows complex rules**
  - **Filenames starting with a dash (-) confuse applications - they interpret the filename as an option. Use a relative path, i.e.:**  

```
# rm ./-i
```
- **The following quoting method works for all filenames:**
  - **Replace all single quotes (') with the sequence '\''**
  - **Add a single quote at the start and end of the filename**
  - **Example:**  

```
# mkdir '/home/Bob'\''s Home Directory'
```
- **In shell scripts, variables containing filenames (or user-supplied input) must always be enclosed in double quotes:**  

```
# for f in *; do wc "$f"; done
```

# Lab Exercise: Special filenames

1. Create the directory "John's Mail Folder" (without the double quotes).
2. Test what happens if you use the filename completion offered by various shells when you try to `cd` to this directory:

```
# ksh
# set -o emacs
# cd John<Esc><Esc>
# exit
```

```
# zsh
# cd John<TAB>
# exit
```

3. Remove the directory.
4. (*Advanced*) Create and delete a file called "`-rf *`". Be careful!

# Shell: Running programs

---

- **A basic command line consists of a command and its arguments.**
  - **Wildcards (such as \* or ?) are expanded by the shell - the program gets the individual filenames as arguments.**
  - **The shell handles input and output redirection:**

```
# ls -l > list_of_files
# make 2>&1 | tee make.log
```
- **Each program returns a numerical *status code*.**
  - **By convention, a return code of 0 means "ok" or "true", anything else indicates an error or "false".**
  - **Display the return code using # echo \$?**
  - **Use logical operators "&&" (and) / "||" (or) to combine multiple commands:**

```
# make && make install
# cd /opt/prog || echo "failed!"
```

# Shell: Process Control

---

- **Most shells offers interactive process control:**
  - **End a command with `&` to run in background**
  - **Use `<Ctrl>-Z` to stop a process**
  - **Send a stopped process into the background with `"bg %1"`**
  - **Kill it with `"kill %1"`**
  - **`jobs` displays a list of background processes**
- **Use `"ps -ef"` to view all processes (`"ps aux"` on BSD-like systems)**
- **The *kill* command can send various signals:**
  - **`-TERM / -15` (default): Kill gracefully (process may do cleanup first)**
  - **`-HUP / -1` (Hangup): often used by server processes to re-read config files**
  - **`-KILL / -9`: kill unconditionally (use as last resort; try `-TERM` first)**

# The Unix toolbox

---

- **The Unix toolbox approach: small programs that do one simple job, and that can be used as building blocks for more complex tasks.**
- **Most tools follow the "no news is good news" approach - don't display unnecessary header or footer lines, and don't generate any output if there is nothing to do.**
- **Use the pipe symbol " | " to send the output of one program to another.**
- **Use backquotes " ` " to substitute the output of a command into the current command line.**

# Toolbox examples

---

- **Count active processes with userid *root*:**

```
# ps -ef | grep '^root' | wc -l
```

- **Display the disk quotas for all users in */etc/passwd*:**

```
# for i in `cut -d: -f1 /etc/passwd`; do quota -u $i; done
```

- **Send a HUP signal to the *inetd* process:**

```
# kill -HUP `ps -ef | grep inetd | grep -v grep | awk '{print $2}'`
```

# The Unix toolbox: Useful Utilities

---

<b>echo</b>	Print a text line
<b>read</b>	Read a line of text from the standard input
<b>grep</b>	Search text for regular expressions - use <code>-v</code> to suppress matched lines, and <code>-i</code> for case-insensitive search
<b>sort</b>	Sort text lines, use <code>-n</code> for numeric sort, and <code>-k</code> for field selection
<b>tee</b>	Save a copy of output in a file
<b>head</b>	Show the first lines of a file
<b>tail</b>	Show the last lines of a file (use <code>-f</code> to watch output in real time)
<b>cut</b>	Select subfields from delimited text
<b>find</b>	Search for files (see separate entry)
<b>wc</b>	Count words, text lines or characters
<b>date</b>	Display the current date. Use <code>'date +%Y-%m-%d_%H-%M-%S'</code> for ISO format.
<b>more</b>	Display text page by page ( <b>less</b> is a more powerful open-source alternative)
<b>cat</b>	Combine multiple files into a single output stream
<b>dd</b>	low-level read/write operations
<b>sed</b>	Stream editor - search+replace on text
<b>awk</b>	Pattern matching based programming language

# *find* - search for files

---

- The *find* utility is a very flexible tool for locating files in the local filesystem.
- The syntax is *find [OPTIONS] DIRS EXPRESSIONS*
  - `-xdev` option: don't cross filesystem boundaries
  - `-follow` option: dereference symlinks
- Examples:
  - Recursive file listing of current directory:  
`# find . -ls`
  - Find file names using wild cards:  
`# find /etc -name '*.conf'`
  - Search for *setuid* or *setgid* binaries:  
`# find / -perm -2000 -o -perm -4000`
  - Search for a string in file contents recursively:  
`# find /etc -type f | xargs grep umask`

# *tar* - archiving tool

---

- The *tar* tool can be used to pack files and directories into an archive file, and extract them again.
  - Create an archive: `# tar cvf archive.tar FILES`
  - View content list: `# tar tvf archive.tar`
  - Extract files: `# tar xvf archive.tar`
- Important options:
  - `-v` : verbose
  - `-p` : preserve permissions when unpacking
  - `-h` : follow symlinks instead of just storing the link
  - `-1` : (GNU tar only) don't cross filesystem boundaries
- Standard *tar* does not handle compression directly.
  - Use a pipeline:  
`zcat COMPRESSED.tar.Z | tar tvf -`  
`tar cvf - DIRECTORY | gzip > COMPRESSED.tar.gz`
  - GNU tar supports `-z/-Z` compression options

# Using *tar* to copy or move directory trees

---

- A common administrative task is copying or moving directory trees across filesystems, while preserving permissions, symlinks and other special files.
- A *tar* pipeline can be used for this task - use the `-p` flag to preserve permissions:

```
# cd /destination  
# ( cd /source; tar cf - . ) | tar xvpf -
```

- If the *netcat* utility (usually installed as `nc`) is available, this can be used to move directories across the network:

```
desthost# cd /destination  
desthost# nc -l -p 7654 | tar xvpf -  
srchost# cd /source  
srchost# tar cf - . | nc desthost 7654
```

# Regular Expressions

---

- Regular expressions are used by many Unix tools for searching and replacing text, including *vi*, *grep*, *sed*, *awk* and *perl*.
- A '\*' is never used by itself, but always attached to a character. Use '.' to match any single character.
  - `a*b` matches any number of 'a' followed by 'b', such as 'b', 'ab', 'aaaaab'
  - `a.*b` matches an 'a' followed by any other characters followed by a 'b', such as 'ab', 'axyzb'
- In Search/Replace operations, use parentheses "(" to record matches, and "\1", "\2" to refer to them in the replacement string.
- Examples:
  - Show active *inetd.conf* entries (not starting with '#'):  
`# grep -v '^#' /etc/inetd.conf`
  - Reverse order of last two words in a file:  
`# sed 's/\([^ ]*\) \([^ ]*$)/\2 \1/' <file`

# Regular Expression Overview

---

<b>^</b>	<b>start of line</b>
<b>\$</b>	<b>end of line</b>
<b>.</b>	<b>any single character (except newline)</b>
<b>.*</b>	<b>zero or more arbitrary characters</b>
<b>+</b>	<b>one or more of the preceding</b>
<b>*</b>	<b>zero or more of the preceding</b>
<b>[0-9]</b>	<b>a single digit</b>
<b>[0-9a-fA-F]+</b>	<b>one or more hex digits</b>
<b>\( \)</b> <b>( )</b>	<b>Grouping (vi, sed)</b> <b>Grouping (perl)</b>
<b>\1, \2, ...</b>	<b>Insert matched group in replacement string</b>

# Scripting basics

---

- **Shell scripts can use the same commands as used interactively.**
- **Use "\$1", "\$2" to refer to the shell script's arguments, or "\$@" to refer to all of them. *Include* the double quotes!**
- **Developing and debugging shell scripts:**
  - **Use || and && for error checking**
  - **Debug shell scripts by running them with "sh -x SCRIPT"**
- **Loops and conditional expressions are supported:**

```
for file in *
do
    echo "Word count for $file:"
    wc "$file"
    if [ -l "$file" ]
    then
        echo "$file is a symlink"
    fi
done
```

# Shell script example: minimal Solaris *top*

---

```
#!/bin/sh -- top
uptime
ps -ef -o user,pcpu,pmem,pid,time,comm | (
  IFS="" read HEADER
  echo "$HEADER"
  sort -r -n -k 2
) | head
```

# Lab Exercise: *watch* shell script

1. Make sure that the `/usr/local/bin/` directory exists:

```
# mkdir -p /usr/local/bin
# cd /usr/local/bin
```

2. Create the script: `# cat >watch`

```
#!/bin/sh -- watch
while true
do
    clear
    echo `date`: "$@"
    echo
    "$@"
    sleep 2
done
```

3. Make it executable with `# chmod +x watch`

4. Test it:

```
# ./watch df -k
# ./watch netstat -n
```

5. (*Advanced*) Add a `head` command to display the first 20 lines only.

# Beyond the toolbox - perl, python

---

- **Shell scripts are not particularly well suited for building complex applications.**
- **High-level languages such as *perl* or *python* offer many advantages:**
  - **Flexible builtin data types (i.e. associative arrays, lists, references) and control structures**
  - **Reliable data processing without arbitrary size or content limitations, i.e. for binary data**
  - **Hundreds of available library modules for common tasks (i.e. system administration, network clients, GUI toolkits)**
  - **Platform independent scripts run on Unix, NT, MacOS, ...**
- **References:**
  - <http://www.perl.org> and <http://www.cpan.org>
  - <http://www.python.org>

# A Perl script example: *rename*

---

- **This script is very useful for renaming multiple files:**

```
# rename 'tr/A-Z/a-z/' * (change to lowercase)
# rename 's/\.htm$/\.html/i' * (rename *.htm to *.html)
# rename 's/\.d+/.sprintf "%03d", $&/e' * (1->001, 23->023, ...)
```

```
#!/usr/bin/env perl

use Getopt::Std;
getopts('n');

$op=shift;
if (!$op || !@ARGV) { die "Usage: rename perlexpr [-n] filenames\n" }

for (@ARGV) {
    $orig = $_;
    eval $op;
    die $@ if $@;
    if ($opt_n) {
        print "mv $orig $_\n" unless $orig eq $_;
    } else {
        rename($orig, $_) unless $orig eq $_;
    }
}
```

# File systems and *mount*

---

- Each filesystem independently allocates the free space:
  - Use `df -k` to view the available disk space for each filesystem
  - A filesystem can run out of inodes if many very small files are stored - use `df -i` to check.
- The root filesystem is mounted by the kernel, the others during the boot process according to the `/etc/fstab` file.
  - `mount` and `umount` can also be used manually.
  - Automounters are available in some cases - i.e., use `volcheck` to mount CD-ROMs on Solaris.
- New filesystems are created with a filesystem-specific version of the `mkfs` command.

# User authentication

---

- **When a user logs in, the system checks the authentication and assigns user and group IDs.**
  - ***/etc/passwd* assigns a numerical user ID and default group for each username. It must be world-readable (i.e., *ls* needs it).**
  - ***/etc/group* assigns additional group memberships.**
  - ***/etc/shadow* contains the encrypted password, and is protected from unauthorized readers.**
- **The Pluggable Authentication Modules (PAM) system allows flexible configuration of authentication systems beyond simple passwords.**
  - ***/etc/pam.d/* contains configuration files for all programs that need to perform user authentication.**
  - **The unified approach ensures consistent results.**

# User administration

---

- The system-dependent *adduser* or *useradd* programs create new user IDs.
  - The required information is added to the */etc/passwd* file.
  - User authentication information is added to */etc/shadow* and possibly other files according to the PAM configuration.
  - A home directory is created, and the contents of */etc/skel* copied into it.
- Refer to the manual page for details.

- ***init* is the process started by the kernel to initialize the user-space bootup process.**
- **The actions taken by *init* are configured in the file */etc/inittab*.**
- ***init* can differentiate between different run levels, and start and stop processes as required.**
  - **On most systems, runlevel 2 is for networked workstation use, and runlevel 3 adds server processes.**
  - **Use `init N` to switch to runlevel N - this often does not work properly because not all software installs the needed start and stop scripts.**
  - **Default is set by the *initdefault* entry in */etc/inittab*.**
- ***init* is also responsible for monitoring and restarting processes that should run permanently:**
  - **getty/login on the local and serial consoles**
  - **other applications, i.e. *daemontools* for djbdns**

# Startup and Shutdown

---

- Shell scripts in `/etc` control the system startup and shutdown process.
- When entering runlevel `N`, first the kill scripts (names start with capital "K") in `/etc/rcN.d` are all run (in ASCII order) with argument "stop", then the start scripts (names start with "S") are run with argument "start".
- The scripts in the `rcN.d` directories should be (symbolic) links to files in the directory `/etc/init.d/`.
- The recommended shutdown command varies according to the operating system:
  - Solaris: use "init 6", or (if that doesn't work) "shutdown -g0 -y"
  - Linux: "shutdown -h now" (-h: halt; use -r for reboot)

- Many network services do not have their own startup procedure, but are instead launched by the *inetd* master server.
- The file */etc/inetd.conf* contains entries for the services that should be started.
  - The port numbers for named services are listed in the file */etc/services*
  - Send the *inetd* process a HUP signal to reread a changed config file:

```
# ps -ef | grep inetd
# kill -HUP PID
```
- Format: "*service type proto flags user server\_path args*", i.e.:  
ftp stream tcp nowait root /usr/sbin/in.ftpd in.ftpd
- For additional security, use the *tcpd* program from the *tcpwrapper* package to control access to services.

# ***cron***

---

- The *cron* daemon is responsible for running background programs at specific times.
- Use the `crontab` command to edit entries:
  - Edit the current user's entries: `# crontab`
  - Edit a different user's entries: `# crontab -l USER`
- A crontab line consists of a time specification (minute, hour, day, month, weekday) and a command.
  - Run a backup every Sunday at 03:10:  
`10 3 * * 0 /usr/local/bin/backup`
  - Run security check every ten minutes:  
`5,15,25,35,45,55 * * * * /usr/local/bin/chk`

- The *syslog* daemon provides basic infrastructure for log message handling.
  - **Priorities:** debug, info, notice, warning, error, critical, alert, emergency
  - **Facilities:** kernel, user, mail, daemon, auth, syslog, lpr, news, uucp, cron, authpriv, ftp, local0-local7
- The file */etc/syslog.conf* controls syslog actions:
  - Different actions can be configured based on the facility and priority.
  - Possible actions include writing the information to a log file, the terminal of a logged-in user, or a remote host.
- Use the *logger* command in scripts to generate syslog entries.  
**Example:**

```
# logger -p auth.err "Intruder Alert!"
```

# Network debugging tools

---

- ***netstat* shows information and statistics related to the network (add `-n` flag to suppress resolving names):**

- **Open TCP connections:** # `netstat`
- **Routing table:** # `netstat -r`
- **Bound (listening) network ports:** # `netstat -a`
- **Interface statistics:** # `netstat -i`

- ***tcpdump* and *snoop* (Solaris) show captured network packets:**

- **Example:**

```
# tcpdump -i eth0 not port 137 and not port 139
```

- ***lsof* shows which process is using a network port:**

- **Who has bound the SMTP port (25/tcp)?**

```
# lsof -i TCP:25
COMMAND PID USER   FD   TYPE DEVICE SIZE NODE NAME
master  548 root   9u   IPv4  560          TCP *:smtp (LISTEN)
```

# Local debugging tools

---

- ***ps* lists processes, *top* shows a realtime display sorted by CPU or memory usage**
- ***truss* (Solaris) or *strace* show all system calls made by a program.**
  - **very useful if a program fails with the message "file not found" and you want to know *which* file it tried to open.**
  - **Use the `-p PID` option to trace a running process.**
  - ***ltrace* (Linux) traces C library calls.**

# Lab Exercise: Using *truss*

1. The object of this exercise is to find out how the *who* command finds the information it prints. First, run "who" to view the listing of logged-in users.

```
# who  
root pts/0 Jun 17 16:44 (10.1.1.72)
```

2. Use *truss* to view the full list of system calls used. Which file(s) does *who* extract its information from? (Hint: look for "open" calls)

```
# truss who
```

3. Limit the displayed system calls:

```
# truss -t open who
```

4. (*Advanced*) How does *who* determine the current time, and how does it know which time zone to use? (Hint: run "set")

# Lab Exercise: Monitoring scripts

1. Goal: generate a syslog entry if someone attempts to connect to the *imap* port (143/tcp), i.e. a port scanner or exploit tool
2. Put the following entry into */etc/inetd.conf*:

```
imap stream tcp nowait nobody /usr/bin/logger logger -p  
auth.err IMAP access!
```

3. Activate the change (search for the *identd* PID using "ps", and send it a HUP signal using "kill")
4. Test the trap:  

```
# telnet localhost 143
```
5. (*Advanced*) Write a shell script that sends mail if the free space in */var/adm* drops below 20%, and run it every hour. (Don't send mail more than once for each occurrence.)

# Network security

---

- **Any service reachable over the network is a potential security risk - unfortunately, Unix default installations usually have many unnecessary services switched on.**
- **Only the minimum required services should be active.**
- **Use `netstat -na`, `ps -ef` and `lsof` to determine which services are active.**

# Local security

---

- **Local security is intended to prevent unprivileged users from gaining additional rights - especially root rights.**
- **Even systems without interactive users should be locally secured.**
  - **If a security hole is exploited in an unprivileged server, the attacker will try to gain root rights.**
- **The most problematic areas are:**
  - **unsafe *setuid* programs**
  - **world-writable files and directories**
  - **unsafe use of files in */tmp***
  - **exploiting trust (root user has *~user/bin* in path)**

# SUN: OpenBoot

---

- **SUN hardware contains the OpenBoot manager that:**
  - **controls the boot process**
  - **enables operating system (re-)installation**
  - **allows entering a debugging mode; often by accident**
- **There are several ways to reach the OpenBoot "ok" prompt:**
  - **On a serial console, send a BREAK signal using your terminal program.**
  - **On the local keyboard, press <Stop>-A.**
  - **Power off a PC or notebook while connected to the serial console.**
- **All processing stops while OpenBoot is active; enter "go" to resume normal operations.**
- **References:**
  - **Sun AnswerBook "OpenBoot Command Reference Manual"**
  - **eeprom command man page**

# SUN: Essential OpenBoot commands

---

<code>go</code>	resume running the operating system
<code>&lt;Ctrl&gt;-?</code>	Display valid command completions (Help)
<code>&lt;Ctrl&gt;-&lt;Space&gt;</code>	Command line completion
<code>boot</code>	Boot kernel from default device
<code>boot cdrom</code>	Boot from CD-ROM
<code>setenv boot-device disk0 disk</code>	Make the first hard disk the default boot device
<code>probe-scsi-all</code>	Show information about all connected SCSI devices
<code>show-disks</code>	Display connected disk drives
<code>devalias</code>	Reconfigure devices
<code>password</code>	Set the EEPROM password - <b>Don't lose it!</b>
<code>setenv security-mode command</code>	Enable asking for EEPROM password when user enters commands or attempts a non-default boot - turn off with <code>setenv security-mode none</code>

# Solaris: Installing the operating system

---

- To install Solaris from a CD-ROM, the system must be in the "OpenBoot prompt" mode and display the "ok" prompt.
- Insert the CD-ROM, and type `boot cdrom - install`
- Follow the prompts - the following basic settings are recommended:
  - Language "English", Locale "USA (ASCII)", terminal "VT100"
- Section 3, "Secure Initial Setup of FireWall-1", contains a step-by-step description of an installation.

# Solaris: Applying operating system patches

---

- Check <http://sunsolve.sun.com> for "Security & Recommended Patches".
- Download the latest patch cluster relevant for the operating system release and architecture.
- Unpack and install the cluster:

```
# unzip 7_Recommended.zip  
# cd PATCH_DIR  
# ./install_cluster
```
- `showrev -p` displays the currently installed patches.

# Overview of software distribution methods

---

- **"tarball" distributions:**
  - consist of a simple "tar" archive (usually compressed) containing the binaries, configuration and other files and directories needed by the program
  - the archive contents are extracted directly into the file system.
- **Software package distributions:**
  - available in many different incompatible varieties (Solaris pkg, RedHat .rpm, Debian .deb, ...)
  - a package contains the software itself and additional information such as installation scripts
  - the package manager keeps track of the installed software.
- **Source code distributions:**
  - require a development environment (C compiler, *make*, ...)
  - The software is configured, compiled, tested and installed.

# Software distribution comparison chart

---

	tar	package	source
<b>Verifies dependencies</b>	-	+	+
<b>Runs install scripts</b>	-	+	+
<b>Preserves existing configuration files</b>	-	+	+
<b>Automatic uninstall and file inventory</b>	-	+	-
<b>Customizable installation paths</b>	-	-	+
<b>Can be installed on machines without compiler or development tools</b>	+	+	-
<b>Platform independent</b>	-	-	+
<b>Integrated into operating system (i.e. automatic startup, log file rotation)</b>	?	+	-
<b>Easily created custom distribution</b>	+	-	+

# Installing tarball distributions

---

- **By convention, distribution tarballs are created relative to the root directory (/).**
  - **If it should be unpacked elsewhere (i.e. in /opt), this should be clearly stated in the accompanying documentation.**
- **The -p (preserve) option must be used when unpacking to ensure that the permission information is preserved.**
  - **Otherwise, some tar implementations apply the current umask, i.e. resulting in a /tmp directory without world write permission**
- **Installation example:**
  1. `# cd /`
  2. `# zcat /stuff/archive.tar.Z | tar tvf - | less`
  3. `# zcat /stuff/archive.tar.Z | tar xvpf -`

# Installing tarball distributions - potential problems

---

- **The archive contents should be examined before unpacking the archive.**
  - **Is the installation location correct?**
  - **Will existing files be overwritten? A one-liner can be used to check for conflicting files:**

```
# tar tf archive.tar | perl -ne 'chomp;  
print "$_ exists!\n" if -f $_'
```
- **Uninstallation is not supported, the files must be deleted manually.**
  - **`tar -tf archive.tar` lists the files and directories contained in the archive**
  - **Warning: be careful not to recursively remove directories (such as `/usr`) that are shared with other programs**

# Creating tarball distributions

---

- A `tar` command line is used to archive the files and directories needed.
  - Usually, the archive should be relative to the root (`/`) directory
  - Directories are stored recursively
- Example:
  1. `# cd /`
  2. `# tar cvf - opt/postfix etc/postfix \`  
`usr/local/bin/postfix | \`  
`gzip > /stuff/archive.tar.gz`

# Software packaging example - Solaris *pkg* format

---

- Adding a package from a Solaris distribution CD-ROM:
  1. mount the CD-ROM with `volcheck`
  2. `# cd /cdrom/cdrom0/s0/Sol*/Prod*`
  3. Add packages from the current directory:  
`# pkgadd -d . SUNWman [...]`
- Adding a package distributed as a single file (e.g. from [www.sunfreeware.com](http://www.sunfreeware.com)):
  1. If the file is compressed, uncompress it  
`gunzip PACKAGE.gz`
  2. `# pkgadd -d PACKAGEFILE`
- Other commands:
  - `pkginfo` : displays a list of installed packages
  - `pkgrm` : removes packages
  - `pkgmk` : create a package (based on a prototype file) that can be installed elsewhere with `pkgadd`

# Solaris *pkg*: Determining file ownership

---

- The `pkgchk` command is intended to verify the integrity of installed packages.
- It can also be used to query which package an installed file belongs to:

```
# echo /usr/bin/truss | pkgchk -l -i/dev/stdin
Pathname: /usr/bin/truss
Type: linked file
Source of link: ../../usr/lib/isaexec
Referenced by the following packages:
    SUNWtoo
Current status: installed
```

# Installing software from a source code distribution

---

- **The unpacked archive usually contains an INSTALL or README file with specific instructions.**
- **Usually, an automated configuration script is provided that:**
  - 1. verifies that the compilation environment works**
  - 2. checks prerequisites such as required libraries**
  - 3. decides which compatibility routines are needed to handle incompatibilities between operating systems**
  - 4. configures installation paths and application options (i.e. supported features or extensions)**
- **Interactive configuration scripts query the user for information, while automatic configuration scripts choose predefined settings, influenced by command line switches or spec files.**

# Commonly used configuration systems

---

- **GNU autoconf (configure and configure.in)**
  1. **check the README and INSTALL files**
  2. **run `./configure --help` and choose the relevant configuration options**
  3. **run the configure command with the required options, for example:**

```
# ./configure --prefix=/opt --without-X11
```
  4. **`# make`**
  5. **`# make install`**
- **Save (and document) the configure command line used.**
  - **This makes it much easier to upgrade the software if a new version is released.**
  - **Use something like the following:**

```
echo './configure ...' > /stuff/configure-PACKAGE
sh /stuff/configure-PACKAGE
```

# Commonly used configuration systems

---

- **Perl modules (Makefile.PL)**
  1. **check README file, and edit Makefile.PL if necessary**
  2. **# perl Makefile.PL**
  3. **# make**
  4. **# make test**
  5. **# make install**
- **multiple predefined Makefiles**
  1. **# cp Makefile.solaris Makefile**
  2. **# make && make install**
- **manual configuration**
  - **edit Makefile, config.h or similar**

# Lab Exercise: Testing the C compiler

- Create a C source file containing a simple program:

1. # mkdir -p /usr/local/src/hworld

2. # cd /usr/local/src/hworld

3. # cat >hello.c

```
#include <stdio.h>
```

```
int main() {
```

```
    puts("Hello World!");
```

```
    return 0;
```

```
}
```

- Compile and test it:

1. # gcc -o hello hello.c

2. # ./hello

# make and Makefiles

---

- **make** is most frequently used to control compilation, but can also be used for other tasks.
- The Makefile contains *targets*, *prerequisites* and *commands*:
  - A target is a file that **make** should keep up to date, i.e. the program executable.
  - The list of prerequisites indicates which other files are needed in order to rebuild the target.
  - If the target is missing or older than one of the prerequisite files, the commands are run to build it.
- **make** automatically handles chains of dependencies, and rebuilds all affected components if a single file is modified.
- **Example Makefile entry:**

```
# target "hello" has prerequisite "hello.c"
hello: hello.c
        gcc -o hello hello.c # rebuild command
```

# More about Makefiles

---

- **Makefiles typically consist of variable definitions (i.e. `INSTDIR=/usr/local`) and rules whose commands use those variables (i.e. `install $(PROG) $(INSTDIR)`)**
- **The command lines in the Makefile must be indented using `<Tab>` characters, not spaces.**
- **Normally, `make` builds the first target listed in the Makefile. An alternative target can be specified; for example, `make install` runs the commands associated with the target named "install".**
- **Most `make` implementations have builtin rules, i.e. how to make a `.o` file out of a `.c` file, that are used in addition to the rules specified in the Makefile.**

# Lab Exercise: Makefile and tarballs

1. Create a Makefile for the "hello world" program and test it.
2. Add a "make test" target that runs the program.
3. Change the greeting message in the C source file and re-run "make test" - the executable should be recompiled automatically.
4. Add a "make install" target to the Makefile that copies the executable to /usr/local/bin.
5. (*Advanced*) Create a README file, and update the "install" target to install it in /usr/local/doc/hworld/.
6. Create a tarball ("hworld-bin.tar.gz") containing the executable (and the README file if you created it).
7. Install the tarball. (*Advanced: Uninstall it.*)
8. (*Advanced*) Add a "make dist" target that automatically creates a source code distribution; then use it to install the program on a different machine.