基本上，在Makefile里会用到install，其他地方会用cp命令。
它们完成同样的任务——拷贝文件，它们之间的区别主要如下：
1、最重要的一点，如果目标文件存在，cp会先清空文件后往里写入新文件，而install则会先删除掉原先的文件然后写入新文件。这是因为往正在 使用的文件中写入内容可能会导致一些问题，比如说写入正在执行的文件可能会失败，比如说往已经在持续写入的文件句柄中写入新文件会产生错误的文件。而使用 install先删除后写入（会生成新的文件句柄）的方式去安装就能避免这些问题了；
2、install命令会恰当地处理文件权限的问题。比如说，install -c会把目标文件的权限设置为rwxr-xr-x；
3、install命令可以打印出更多更合适的debug信息，还会自动处理SElinux上下文的问题。

转：http://blog.csdn.NET/stevenliyong/article/details/4663583

install  - copy files and set attributes
install 在做拷贝的同时，设置attributes.

因此Makefile 中尽量使用install 命令。

例如
```
@install  -d /usr/bin
@install -p -D -m 0755 targets /usr/bin
```

相当于
```
@mkdir -p /usr/bin
@cp targets /usr/bin
@chmod 755 /usr/bin/targets
@touch /usr/bin/tagets                <---- 更新文件时间戳
```

install 命令好强大啊。
另外@前缀的意思是不在控制台输出结果。

转载:http://www.cnblogs.com/wwwsinagogogo/archive/2011/08/15/2139124.html

【概述】
Install和cp类似，都可以将文件/目录拷贝到指定的地点。但是，install允许你控制目标文件的属性。install通常用于程序的makefile，使用它来将程序拷贝到目标（安装）目录。
【语法】
```
install [OPTION]... [-T] SOURCE DEST
install [OPTION]... SOURCE... DIRECTORY
```

install [OPTION]... -t DIRECTORY SOURCE...
install [OPTION]... -d DIRECTORY...
*如果指定了两个文件名，`install’将第一个文件拷贝到第二个
* 如果使用了 `--target-directory’（`-t’）选项，或者如果最后一个文件是一个目录并且没有使用`--no-target-directory’（`-T’)选项， `install’将每一个源文件拷贝到指定的目录，目标文件名与SOURCE文件名相同。
* 如果使用了 `--directory’（`-d’）选项，`install’将逐级创建缺失的目标目录
【常用选项】
-s:对待拷贝的可执行文件进行strip操作，取出文件中的符号表。（一般在做成nand rom时去除符号表，NFS时为了调试方便，一般不会使用此选项）
-d(--directory)：创建制定的目录结构（逐级创建）。如，指定安装位置为/usr/local/aaa/bbb，/usr/loacal已存在，install会帮助我们创建aaa和bbb目录，并把程序安装到指定位置

If we hand write a Makefile, we should always stick to install instead of using cp for the installation commands. Not only is it more convenient, but it does things right (cp does things *wrong*).
For example, if we attempt to update /bin/bash, which is currently running, with "cp ... /bin/bash", we get a "text busy" error. If we attempt to update /lib/libc.so.6 with "cp ... /lib/libc.so.6", then we either get "text busy" (in ancient versions of **Linux**) or breaks each and every running program within a fraction of a second (in recent versions of Linux). install does the thing right in both situations.
The reason why cp fails is that it simply attempts to open the destination file in write-only mode and write the new contents. This causes problem because Linux (and all contemporary Unices as well as Microsoft Windows) uses memory mapping (mmap) to load executables and dynamic libraries.
The contents of an executable or dynamic library are mmap'd into the linear address space of relevant processes. Therefore, any change in the underlying file affects the mmap'd memory regions and can potentially break programs. (MAP_PRIVATE guarantees changes by processes to those memory regions are handled by COWwithout affecting the underlying file. On the contrary, POSIX leaves to implementations whether COW should be used if the underlying file is modified. In fact, for purpose of efficiency, in Linux, such modifications are *visible* to processes even though MAP_PRIVATE may have be used.)
There is an option MAP_DENWRITE which disallows any modification to the underlying file, designed to avoid situations described above. Executables and dynamic libraries are all mmap'd with this option. Unfortunately, it turned out MAP_DENYWRITE became a source of DoS attacks, forcing Linux to ignore this option in recent versions.
Executables are mmap'd by the kernel (in the execve syscall). For kernel codes, MAP_DENYWRITE still works, and therefore we get "text busy" errors if we attempt to modify the executable.
On the other hand, dynamic libraries are mmap'd by userspace codes (for example, by loaders like /lib/ld-linux.so). These codes still pass MAP_DENYWRITE to the kernel, but newer kernels silently ignores this option. The bad consequence is that you can break the whole system if you think you're only upgrading the C runtime library.

Then, how does `install` solve this problem? Very simple – unlinking the file before writing the new one. Then the old file (no longer present in directory entries but still in disk until the last program referring to it exits) and the new file have different inodes. Programs started before the upgrading (continuing using the old file) and those after the upgrading (using the new version) will both be happy.

记得在大学的时候在编译LFS 6 的时候, 一直搞不懂 install 的命令 和 cp 以及和 chmod, chgrp 的区别?

工作之后才明白一个Running 的进程不能随便进行 cp , 经常会提示 "text busy " , 运维部的前辈们给的建议是采用mv 来替代 cp , 今天看起来前辈好像不知道install 这个命令啊.

今天就简单介绍一下 install 命令.

install copy 文件列表且同时能够设置文件的属性(包括 owner, group) , 通常用在 Makefiles 中 用来copy 程序到指定的目录.

常见的用法有以下3中形式:
1: install -d [option] DIRECTORY [DIRECTORY...] 支持多个. 类似 mkdir -p 支持递归.
例如: install -d a/b/c e/f 结果和 mkdir -p a/b/c e/f 一样.
2: install [option] SOURCE DEST
复制 SOURCE 文件(测试不能是目录) 到DEST file(文件) .
install a/e c 结果类似 cp a/e c # 注意c必须是文件.
有用选项 -D
install -D x a/b/c # 效果类似 mkdir -p a/b && cp x a/b/c
3: install [option] SOURCE [SOURCE...] DIRECTORY
复制 多个SOURCE 文件到目的目录.
install a/* d 其中 d 是目录.

有用选项
-b： 自动备份.
-m : 设置安装文件的权限
-p :保留文件的timestamps. 也就是说文件的timestaamps 和 source 文件一样. 当我们想要利用安装文件的mtime来跟踪文件的build时间而不是 安装时间.
-s : Strip the symbol tables from installed binary executables.
-S : 备份文件的后缀.

install  -S .bak new  old  #old 文件自动被 mv 为 old.bak.
-v: verbose ,打印install 的文件的详细信息.
`-c'
    Ignored; for compatibility with old Unix versions of `install'.  #用来兼容旧版的unix.

-C: (大写)
安装文件, 但是如果目标文件和源文件一样( 判断方法需要看看代码确认) 就跳过, 这样的好处是 能够保持一样文件的mtime.