# Disk and File Utilities

ComS 252 — Iowa State University

Andrew Miner

# /dev

- Recall: in UNIX, `/dev` is a directory for devices

Character device : can read/write a character at a time

- Like the keyboard

Block device : can read/write a block of characters at a time

- Like a disk

# /dev

- Recall: in UNIX, /dev is a directory for devices

Character device : can read/write a character at a time

- Like the keyboard

Block device : can read/write a block of characters at a time

- Like a disk

- Ok, so what's happening in /dev?

# /dev

- Recall: in UNIX, /dev is a directory for devices

Character device : can read/write a character at a time

- Like the keyboard

Block device : can read/write a block of characters at a time

- Like a disk

- Ok, so what's happening in /dev?
- In UNIX, devices act like files

# /dev

- Recall: in UNIX, /dev is a directory for devices

Character device : can read/write a character at a time

- Like the keyboard

Block device : can read/write a block of characters at a time

- Like a disk

- Ok, so what's happening in /dev?
- In UNIX, devices act like files
- There are virtual files in /dev, that are actually devices
  - Write to a file → send data to a device
  - Read from a file → read data from a device

# Some important devices

`/dev/null` : Null device

- Ignores all data written to it
- Produces nothing

`/dev/zero` : Zero device

- Provides null (zero) characters, forever

`/dev/random` : random stream

- Blocking: will wait if the stream is exhausted

`/dev/urandom` : random stream

- Non-Blocking

`/dev/tty`*n* : terminals

`/dev/audio` : sound — old school

`/dev/dsp` : digital sampling device

# Drives in Linux

`/dev/hda` : First IDE drive

`/dev/hdb` : Second IDE drive

`/dev/hdc` : Third IDE drive

`/dev/hdd` : Fourth IDE drive

`/dev/sda` : First SATA or SCSI drive

`/dev/sdb` : Second SATA or SCSI drive

⋮

To access a particular partition, append the partition number; e.g.,

`/dev/hda1` : First IDE drive, partition 1

`/dev/hdc5` : Third IDE drive, partition 5

`/dev/sdb4` : Second SATA or SCSI drive, partition 4

# Partitioning tools

## fdisk

- Just like the old MS-DOS utility
- Not the easiest to use, but gets the job done
- Need to specify the device to partition:

```
prompt# fdisk /dev/hdb
```

## parted

- GNU partition editor; "smarter" than `fdisk`
- Can handle GUID partition tables
- If you don't specify a device, it will guess

# Logical Volume management

## `lvm`

- Large tool for all kinds of logical volume administration
- Default gives you an interactive "lvm shell"
- Or, use the following utilities for specific actions:

| | | |
|---|---|---|
| lvs | : | Report information about logical volumes |
| lvcreate | : | Create a logical volume |
| lvremove | : | Remove a logical volume |
| lvresize | : | Resize a logical volume |
| vgs | : | Report information about volume groups |
| vgcreate | : | Create a volume group |
| vgremove | : | Remove a volume group |

- The above "utilities" simply invoke `lvm` appropriately

# Formatting and such

## mkfs

- Builds a filesystem ("formats a disk")
- You need to specify the device (usually, a partition)
  - Or you can specify a file and a size to build a "disk image"
  - But: use `mkisofs` instead, to build an ISO image

`-t` : Specify the filesystem type (or get the default)

```
prompt# mkfs -t vfat /dev/hdb3
```

## resize2fs

- Resize an ext2, ext3, or ext4 file system
- Use this before or after changing partitions or LVs

# Mounting and unmounting filesystems

## `mount`: Add a device to the filesystem tree

- Specify the device, and the mount point

`-t` : Specify the filesystem type (`mount` can sometimes guess)

```
prompt# mount -t vfat /dev/hdb2 /Win/D
```

## `umount`: remove a device from the filesystem tree

- Only need to specify the mount point, or the device
- Will fail if the device is busy

```
prompt# umount /Win/D
```

# Automatic mounting (1)

What if I have a device that

1. is always connected
2. is always mounted at the same point

Do I have to manually `mount` it every time I boot?

# Automatic mounting (1)

What if I have a device that

1. is always connected
2. is always mounted at the same point

Do I have to manually `mount` it every time I boot? No.

## /etc/fstab

- Configuration file; can edit in any text editor
- One entry per line:
  1. The device
  2. The mount point
  3. Filesystem type
  4. Other mount options (comma separated list)
  5. For backups; use 0 for none
  6. For file system checks
- `man fstab` for more details

# Automatic mounting (2)

- Items in `/etc/fstab` are mounted at boot time
  - . . . except for those specifying "`noauto`" in the options
- Items in `/etc/fstab` can be mounted manually
  - Only need to specify device or mount point, e.g:

```
prompt# mount /Win/D
```

# Automatic mounting (2)

- Items in `/etc/fstab` are mounted at boot time
  - . . . except for those specifying "`noauto`" in the options
- Items in `/etc/fstab` can be mounted manually
  - Only need to specify device or mount point, e.g:

```
prompt# mount /Win/D
```

What about devices that are not always connected?

# Automatic mounting (2)

- Items in `/etc/fstab` are mounted at boot time
  - . . . except for those specifying "`noauto`" in the options
- Items in `/etc/fstab` can be mounted manually
  - Only need to specify device or mount point, e.g:

```
prompt# mount /Win/D
```

What about devices that are not always connected?

- Mount attempts may hang if the device is not present

# Automatic mounting (2)

- Items in `/etc/fstab` are mounted at boot time
  - . . . except for those specifying "`noauto`" in the options
- Items in `/etc/fstab` can be mounted manually
  - Only need to specify device or mount point, e.g:

```
prompt# mount /Win/D
```

What about devices that are not always connected?

- Mount attempts may hang if the device is not present
- Must use `noauto` if there is an entry in `/etc/fstab`

# Automatic mounting (2)

- Items in `/etc/fstab` are mounted at boot time
  - . . . except for those specifying "`noauto`" in the options
- Items in `/etc/fstab` can be mounted manually
  - Only need to specify device or mount point, e.g:

```
prompt# mount /Win/D
```

What about devices that are not always connected?

- Mount attempts may hang if the device is not present
- Must use `noauto` if there is an entry in `/etc/fstab`
- Must manually `mount` and `umount` the device

# Automatic mounting (2)

- Items in `/etc/fstab` are mounted at boot time
  - . . . except for those specifying "`noauto`" in the options
- Items in `/etc/fstab` can be mounted manually
  - Only need to specify device or mount point, e.g:

```
prompt# mount /Win/D
```

What about devices that are not always connected?

- Mount attempts may hang if the device is not present
- Must use `noauto` if there is an entry in `/etc/fstab`
- Must manually `mount` and `umount` the device
- . . . or, use `automount`
  - We will discuss this later

# Example /etc/fstab

```
/dev/mapper/vg_mbp13-lv01   /          ext4     defaults   1   1
/dev/sda5                   /boot      ext4     defaults   1   2
/dev/sda4                   /boot/efi  hfsplus  defaults   0   2
/dev/sda7                   /home      ext4     defaults   1   2
/dev/mapper/vg_mbp13-lv00   swap       swap     defaults   0   0
/dev/sda2                   /mnt/Mac   hfsplus  defaults   0   0
/dev/sda3                   /mnt/Win   ntfs     defaults   0   0
```

Almost actual file, on a triple–boot Macintosh

# Disk usage

## `df`

- For each mounted filesystem, shows
  - The space used
  - The space available

`-h` : Use "human readable" sizes

- E.g., "1.2G" instead of "1288490189"

# Example df

```
prompt$ df -h
Filesystem      Size   Used   Avail   Use%   Mounted on
rootfs           34G   3.0G     31G     9%   /
devtmpfs        3.8G      0    3.8G     0%   /dev
tmpfs           3.8G      0    3.8G     0%   /dev/shm
tmpfs           3.8G   936K    3.8G     1%   /run
tmpfs           3.8G      0    3.8G     0%   /sys/fs/cgroup
tmpfs           3.8G      0    3.8G     0%   /media
/dev/sda3        48G    26G     23G    53%   /mnt/Win
/dev/sda2       355G    66G    290G    19%   /mnt/Mac
/dev/sda7        20G   824M     18G     5%   /home
/dev/sda5       497M    75M    398M    16%   /boot
/dev/sda4       128M   4.1M    124M     4%   /boot/efi
prompt$
```

Almost actual output, on a triple–boot Macintosh

# Links

Question: can one file have two different path names?

# Links

Question: can one file have two different path names?

Of course, I can do things like

```
/home/alice/foo.txt
~/foo.txt
/home/alice/../alice/foo.txt
```

# Links

Question: can one file have two different path names?

Of course, I can do things like

```
/home/alice/foo.txt
~/foo.txt
/home/alice/../alice/foo.txt
```

But what if we don't use relative path tricks. Can the path names

```
/home/alice/foo.txt
```

```
/home/alice/Notes/bar.txt
```

refer to the same file?

# Links

Question: can one file have two different path names?

Of course, I can do things like

```
/home/alice/foo.txt
~/foo.txt
/home/alice/../alice/foo.txt
```

But what if we don't use relative path tricks. Can the path names

```
/home/alice/foo.txt
```

```
/home/alice/Notes/bar.txt
```

refer to the same file?

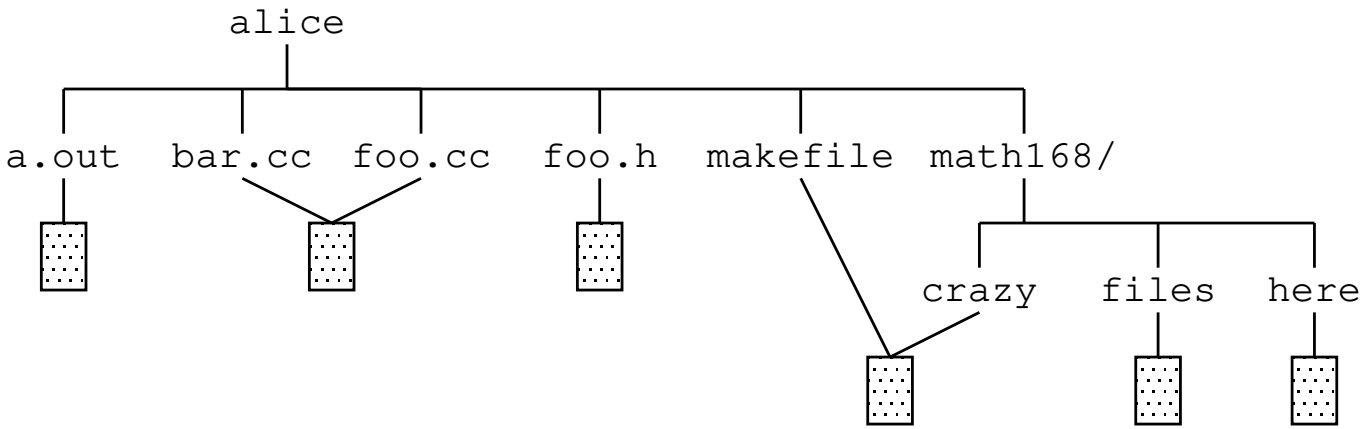
Answer: YES (depending on the filesystem type)

Two different ways to do this:

1. "Hard" links
2. "Soft" or "symbolic" links

# Hard Links

- Different names for the same file
  - There are multiple directory entries
  - The entries "point" to the same file
  - Can be across directories




- Must be within a filesystem (the same partition)
- To set this up: use `ln`
  - Syntax is identical to `cp`
- Fun fact: directories `.` and `..` are hard links

# Example: hard links

```
prompt$
```

# Example: hard links

```
prompt$ ls
```

# Example: hard links

```
prompt$ ls
foo.txt
prompt$
```

# Example: hard links

```
prompt$ ls
foo.txt
prompt$ ln foo.txt bar.txt
```

# Example: hard links

```
prompt$ ls
foo.txt
prompt$ ln foo.txt bar.txt
prompt$
```

# Example: hard links

```
prompt$ ls
foo.txt
prompt$ ln foo.txt bar.txt
prompt$ ls -l
```

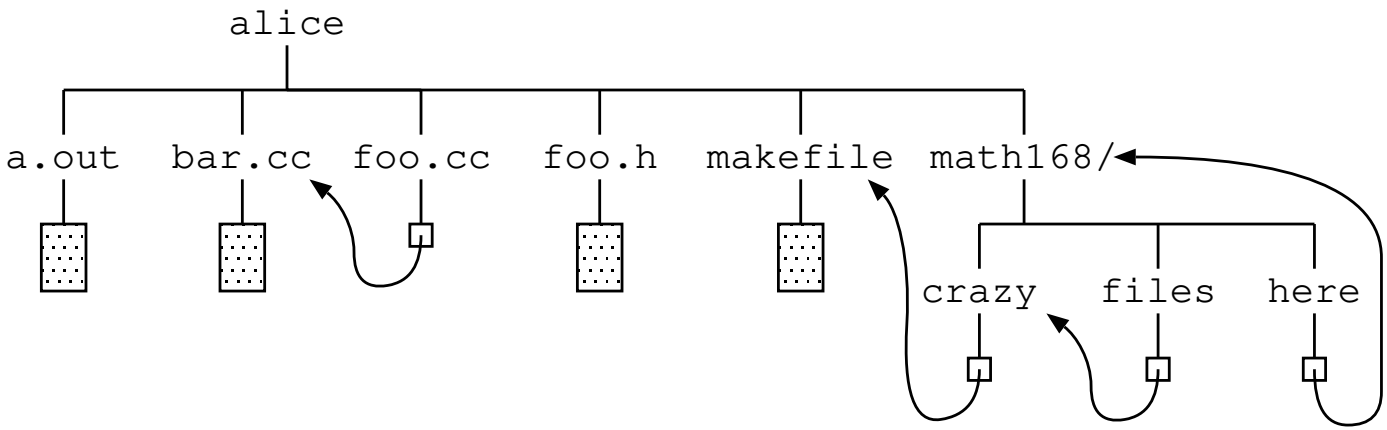
# Example: hard links

```
prompt$ ls
foo.txt
prompt$ ln foo.txt bar.txt
prompt$ ls -l
total 42
-rw-r----- 2 alice hackers 20935 Jul 11 15:21 bar.txt
-rw-r----- 2 alice hackers 20935 Jul 11 15:21 foo.txt
prompt$
```

- The second column of the long listing: link count
  - Number of incoming links to the file
- The entries for `bar.txt` and `foo.txt` will remain identical
  - Except for the names

# Symbolic Links

- Like "shortcuts" in NTFS
- A tiny, special file that points to another file or directory
- Can be across directories and across filesystems




- To set this up: use `ln -s`

# Example: symbolic links

```
prompt$
```

# Example: symbolic links

```
prompt$ ls
```

# Example: symbolic links

```
prompt$ ls
foo.txt
prompt$
```

# Example: symbolic links

```
prompt$ ls
foo.txt
prompt$ ln -s foo.txt bar.txt
```

# Example: symbolic links

```
prompt$ ls
foo.txt
prompt$ ln -s foo.txt bar.txt
prompt$
```

# Example: symbolic links

```
prompt$ ls
foo.txt
prompt$ ln -s foo.txt bar.txt
prompt$ ln -s . loop
```

# Example: symbolic links

```
prompt$ ls
foo.txt
prompt$ ln -s foo.txt bar.txt
prompt$ ln -s . loop
prompt$
```

# Example: symbolic links

```
prompt$ ls
foo.txt
prompt$ ln -s foo.txt bar.txt
prompt$ ln -s . loop
prompt$ ls -F
```

# Example: symbolic links

```
prompt$ ls
foo.txt
prompt$ ln -s foo.txt bar.txt
prompt$ ln -s . loop
prompt$ ls -F
bar.txt@    foo.txt    loop@
prompt$
```

# Example: symbolic links

```
prompt$ ls
foo.txt
prompt$ ln -s foo.txt bar.txt
prompt$ ln -s . loop
prompt$ ls -F
bar.txt@    foo.txt    loop@
prompt$ ls -l
```

# Example: symbolic links

```
prompt$ ls
foo.txt
prompt$ ln -s foo.txt bar.txt
prompt$ ln -s . loop
prompt$ ls -F
bar.txt@    foo.txt    loop@
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 bar.txt -> foo.txt
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 foo.txt
lrwxrwxrwx 1 alice hackers     1 Sep 14 12:03 loop -> .
prompt$
```

# Example: symbolic links

```
prompt$ ls
foo.txt
prompt$ ln -s foo.txt bar.txt
prompt$ ln -s . loop
prompt$ ls -F
bar.txt@    foo.txt    loop@
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 bar.txt -> foo.txt
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 foo.txt
lrwxrwxrwx 1 alice hackers     1 Sep 14 12:03 loop -> .
prompt$ cd loop
```

# Example: symbolic links

```
prompt$ ls
foo.txt
prompt$ ln -s foo.txt bar.txt
prompt$ ln -s . loop
prompt$ ls -F
bar.txt@    foo.txt    loop@
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 bar.txt -> foo.txt
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 foo.txt
lrwxrwxrwx 1 alice hackers     1 Sep 14 12:03 loop -> .
prompt$ cd loop
prompt$
```

# Example: symbolic links

```
prompt$ ls
foo.txt
prompt$ ln -s foo.txt bar.txt
prompt$ ln -s . loop
prompt$ ls -F
bar.txt@    foo.txt    loop@
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 bar.txt -> foo.txt
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 foo.txt
lrwxrwxrwx 1 alice hackers     1 Sep 14 12:03 loop -> .
prompt$ cd loop
prompt$ ls
```

# Example: symbolic links

```
prompt$ ls
foo.txt
prompt$ ln -s foo.txt bar.txt
prompt$ ln -s . loop
prompt$ ls -F
bar.txt@    foo.txt    loop@
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 bar.txt -> foo.txt
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 foo.txt
lrwxrwxrwx 1 alice hackers     1 Sep 14 12:03 loop -> .
prompt$ cd loop
prompt$ ls
bar.txt    foo.txt    loop
prompt$
```

# Example: symbolic links

```
prompt$ ls
foo.txt
prompt$ ln -s foo.txt bar.txt
prompt$ ln -s . loop
prompt$ ls -F
bar.txt@    foo.txt    loop@
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 bar.txt -> foo.txt
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 foo.txt
lrwxrwxrwx 1 alice hackers     1 Sep 14 12:03 loop -> .
prompt$ cd loop
prompt$ ls
bar.txt    foo.txt    loop
prompt$ pwd
```

# Example: symbolic links

```
prompt$ ls
foo.txt
prompt$ ln -s foo.txt bar.txt
prompt$ ln -s . loop
prompt$ ls -F
bar.txt@    foo.txt    loop@
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 bar.txt -> foo.txt
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 foo.txt
lrwxrwxrwx 1 alice hackers     1 Sep 14 12:03 loop -> .
prompt$ cd loop
prompt$ ls
bar.txt    foo.txt    loop
prompt$ pwd
/home/alice/Test/loop
prompt$
```

# Example: symbolic links

```
prompt$ ls
foo.txt
prompt$ ln -s foo.txt bar.txt
prompt$ ln -s . loop
prompt$ ls -F
bar.txt@    foo.txt    loop@
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 bar.txt -> foo.txt
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 foo.txt
lrwxrwxrwx 1 alice hackers     1 Sep 14 12:03 loop -> .
prompt$ cd loop
prompt$ ls
bar.txt    foo.txt    loop
prompt$ pwd
/home/alice/Test/loop
prompt$ cd loop/loop/loop
```

# Example: symbolic links

```
prompt$ ls
foo.txt
prompt$ ln -s foo.txt bar.txt
prompt$ ln -s . loop
prompt$ ls -F
bar.txt@    foo.txt    loop@
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 bar.txt -> foo.txt
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 foo.txt
lrwxrwxrwx 1 alice hackers     1 Sep 14 12:03 loop -> .
prompt$ cd loop
prompt$ ls
bar.txt    foo.txt    loop
prompt$ pwd
/home/alice/Test/loop
prompt$ cd loop/loop/loop
prompt$
```

# Example: symbolic links

```
prompt$ ls
foo.txt
prompt$ ln -s foo.txt bar.txt
prompt$ ln -s . loop
prompt$ ls -F
bar.txt@    foo.txt    loop@
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 bar.txt -> foo.txt
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 foo.txt
lrwxrwxrwx 1 alice hackers     1 Sep 14 12:03 loop -> .
prompt$ cd loop
prompt$ ls
bar.txt    foo.txt    loop
prompt$ pwd
/home/alice/Test/loop
prompt$ cd loop/loop/loop
prompt$ ls
```

# Example: symbolic links

```
prompt$ ls
foo.txt
prompt$ ln -s foo.txt bar.txt
prompt$ ln -s . loop
prompt$ ls -F
bar.txt@    foo.txt    loop@
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 bar.txt -> foo.txt
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 foo.txt
lrwxrwxrwx 1 alice hackers     1 Sep 14 12:03 loop -> .
prompt$ cd loop
prompt$ ls
bar.txt    foo.txt    loop
prompt$ pwd
/home/alice/Test/loop
prompt$ cd loop/loop/loop
prompt$ ls
bar.txt    foo.txt    loop
prompt$
```

# Example: symbolic links

```
prompt$ ls
foo.txt
prompt$ ln -s foo.txt bar.txt
prompt$ ln -s . loop
prompt$ ls -F
bar.txt@    foo.txt    loop@
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 bar.txt -> foo.txt
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 foo.txt
lrwxrwxrwx 1 alice hackers     1 Sep 14 12:03 loop -> .
prompt$ cd loop
prompt$ ls
bar.txt    foo.txt    loop
prompt$ pwd
/home/alice/Test/loop
prompt$ cd loop/loop/loop
prompt$ ls
bar.txt    foo.txt    loop
prompt$ pwd
```

# Example: symbolic links

```
prompt$ ls
foo.txt
prompt$ ln -s foo.txt bar.txt
prompt$ ln -s . loop
prompt$ ls -F
bar.txt@    foo.txt    loop@
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 bar.txt -> foo.txt
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 foo.txt
lrwxrwxrwx 1 alice hackers     1 Sep 14 12:03 loop -> .
prompt$ cd loop
prompt$ ls
bar.txt    foo.txt    loop
prompt$ pwd
/home/alice/Test/loop
prompt$ cd loop/loop/loop
prompt$ ls
bar.txt    foo.txt    loop
prompt$ pwd
/home/alice/Test/loop/loop/loop/loop
prompt$
```

# Removing links: use `rm`

## Hard links

- When you `rm` a file, its link count decreases by one
- The file is removed when the link count becomes zero

## Symbolic links

- When you `rm` a symlink, the special file goes away
- If you remove or move the target of a symlink
  - You get a broken link
  - Will cause interesting error messages

# Example: broken links

```
prompt$
```

# Example: broken links

```
prompt$ mv foo.txt oops.txt
```

# Example: broken links

```
prompt$ mv foo.txt oops.txt
prompt$
```

# Example: broken links

```
prompt$ mv foo.txt oops.txt
prompt$ ls -l
```

# Example: broken links

```
prompt$ mv foo.txt oops.txt
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 bar.txt -> foo.txt
lrwxrwxrwx 1 alice hackers     1 Sep 14 12:03 loop -> .
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 oops.txt
prompt$
```

# Example: broken links

```
prompt$ mv foo.txt oops.txt
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 bar.txt -> foo.txt
lrwxrwxrwx 1 alice hackers     1 Sep 14 12:03 loop -> .
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 oops.txt
prompt$ cat bar.txt
```

# Example: broken links

```
prompt$ mv foo.txt oops.txt
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 bar.txt -> foo.txt
lrwxrwxrwx 1 alice hackers     1 Sep 14 12:03 loop -> .
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 oops.txt
prompt$ cat bar.txt
cat: bar.txt: No such file or directory
prompt$
```

# Example: broken links

```
prompt$ mv foo.txt oops.txt
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 bar.txt -> foo.txt
lrwxrwxrwx 1 alice hackers     1 Sep 14 12:03 loop -> .
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 oops.txt
prompt$ cat bar.txt
cat: bar.txt: No such file or directory
prompt$ rm loop
```

# Example: broken links

```
prompt$ mv foo.txt oops.txt
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 bar.txt -> foo.txt
lrwxrwxrwx 1 alice hackers     1 Sep 14 12:03 loop -> .
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 oops.txt
prompt$ cat bar.txt
cat: bar.txt: No such file or directory
prompt$ rm loop
prompt$
```

# Example: broken links

```
prompt$ mv foo.txt oops.txt
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 bar.txt -> foo.txt
lrwxrwxrwx 1 alice hackers     1 Sep 14 12:03 loop -> .
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 oops.txt
prompt$ cat bar.txt
cat: bar.txt: No such file or directory
prompt$ rm loop
prompt$ mv bar.txt foo.txt
```

# Example: broken links

```
prompt$ mv foo.txt oops.txt
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 bar.txt -> foo.txt
lrwxrwxrwx 1 alice hackers     1 Sep 14 12:03 loop -> .
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 oops.txt
prompt$ cat bar.txt
cat: bar.txt: No such file or directory
prompt$ rm loop
prompt$ mv bar.txt foo.txt
prompt$
```

# Example: broken links

```
prompt$ mv foo.txt oops.txt
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 bar.txt -> foo.txt
lrwxrwxrwx 1 alice hackers     1 Sep 14 12:03 loop -> .
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 oops.txt
prompt$ cat bar.txt
cat: bar.txt: No such file or directory
prompt$ rm loop
prompt$ mv bar.txt foo.txt
prompt$ ls -l
```

# Example: broken links

```
prompt$ mv foo.txt oops.txt
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 bar.txt -> foo.txt
lrwxrwxrwx 1 alice hackers     1 Sep 14 12:03 loop -> .
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 oops.txt
prompt$ cat bar.txt
cat: bar.txt: No such file or directory
prompt$ rm loop
prompt$ mv bar.txt foo.txt
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 foo.txt -> foo.txt
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 oops.txt
prompt$
```

# Example: broken links

```
prompt$ mv foo.txt oops.txt
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 bar.txt -> foo.txt
lrwxrwxrwx 1 alice hackers     1 Sep 14 12:03 loop -> .
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 oops.txt
prompt$ cat bar.txt
cat: bar.txt: No such file or directory
prompt$ rm loop
prompt$ mv bar.txt foo.txt
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 foo.txt -> foo.txt
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 oops.txt
prompt$ cat foo.txt
```

# Example: broken links

```
prompt$ mv foo.txt oops.txt
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 bar.txt -> foo.txt
lrwxrwxrwx 1 alice hackers     1 Sep 14 12:03 loop -> .
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 oops.txt
prompt$ cat bar.txt
cat: bar.txt: No such file or directory
prompt$ rm loop
prompt$ mv bar.txt foo.txt
prompt$ ls -l
total 42
lrwxrwxrwx 1 alice hackers     7 Sep 14 12:03 foo.txt -> foo.txt
-rw-r----- 1 alice hackers 20935 Jul 11 15:21 oops.txt
prompt$ cat foo.txt
cat: foo.txt: Too many levels of symbolic links
prompt$
```

# File times in UNIX

A file has 3 times associated with it

1. Modification time: when its contents changed
2. Status time: when its group, owner, permissions changed
3. Access time: when it was last read

- `ls -l`: gives modification time by default
- `ls -cl`: show status time
- `ls -ul`: show access time

# Changing the file time

## `touch`: change file time

- Usage: `touch file1 ... filen`
- Default: sets modification and access times to "now"
- Will create empty files for any that do not exist
- Can set an arbitrary[1] time in the future or past
- Check the `man` page for more details

[1] Within limits, see the next slides

# Unix epoch

File times are stored

- as the number of seconds since the UNIX epoch
  - January 1, 1970, midnight
- using UTC (Coordinated Universal Time)
  - Within fractions of a second of GMT
  - No issues with time zone changes, daylight savings, leap year...

Examples:

- 1,234,567,890 seconds since the Unix epoch:
  - $1234567890/(365 \cdot 24 \cdot 60 \cdot 60) \approx 39.1$ years after Jan 1, 1970

# Unix epoch

File times are stored

- as the number of seconds since the UNIX epoch
  - January 1, 1970, midnight
- using UTC (Coordinated Universal Time)
  - Within fractions of a second of GMT
  - No issues with time zone changes, daylight savings, leap year...

Examples:

- 1,234,567,890 seconds since the Unix epoch:
  - $1234567890/(365 \cdot 24 \cdot 60 \cdot 60) \approx 39.1$ years after Jan 1, 1970
  - More precisely, Feb 13, 2009 at 23:31:30 (UTC)

# Unix epoch

File times are stored

- as the number of seconds since the UNIX epoch
  - January 1, 1970, midnight
- using UTC (Coordinated Universal Time)
  - Within fractions of a second of GMT
  - No issues with time zone changes, daylight savings, leap year...

Examples:

- 1,234,567,890 seconds since the Unix epoch:
  - $1234567890/(365 \cdot 24 \cdot 60 \cdot 60) \approx 39.1$ years after Jan 1, 1970
  - More precisely, Feb 13, 2009 at 23:31:30 (UTC)
    - Apparently, there were parties
- -60 seconds since the Unix epoch:
  - Dec 31, 1969 at 23:59:00 (UTC)

# Y2038 problem

## If we store times using 32–bit signed integers

- Legal times are between $-2^{31}$ and $2^{31} - 1$ seconds since epoch
- $2^{31}$ seconds $\approx$ 68 years
- Range is Unix epoch $\pm$ 68 years
- Precise limits: between 1901 – 12 – 13 and 2038 – 01 –19
- For UNIX, there was no Y2k problem — it's Y2038

# Y2038 potential solutions

## If we store times using 64–bit signed integers

- Legal times are between $-2^{63}$ and $2^{63} - 1$ seconds since epoch
- Range is Unix epoch $\pm$ 293 billion years
- Creationists: universe is 6 – 7 thousand years old
- Big bang theorists: universe is 14 – 20 billion years old
- Either way, this solution is a bit overkill

## If we use 64 bits and store milliseconds since epoch

- Range is Unix epoch $\pm$ 293 million years
- Get better file time resolution
- Do you really need files as old as the universe?

# Summary of today's commands

`df` : show disk space available on devices

`fdisk` : disk partitioning

`ln` : link files

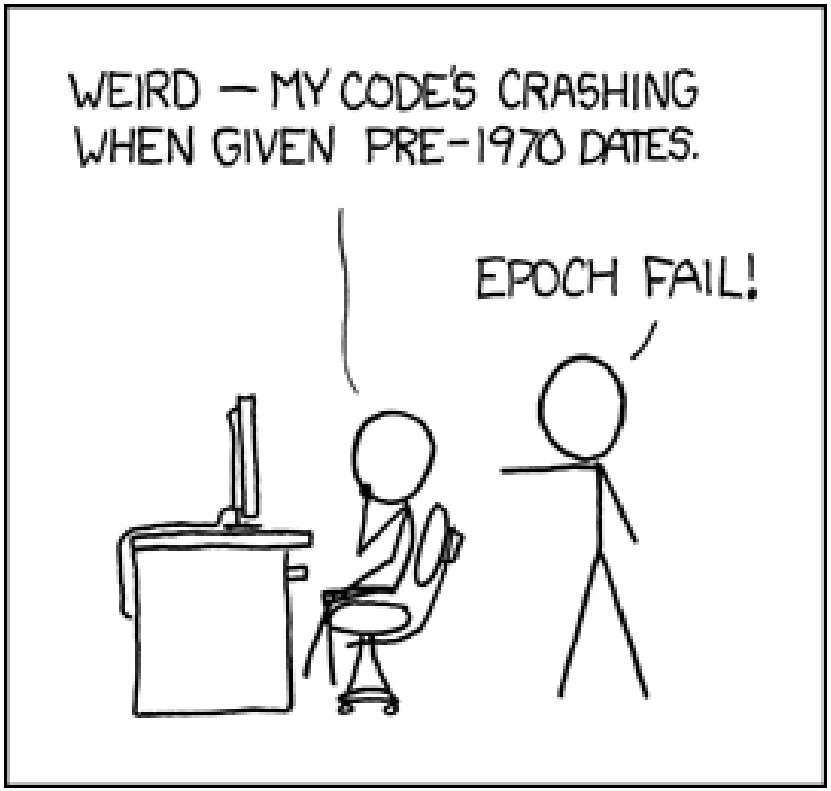
`mkfs` : create a filesystem ("format a disk")

`mount` : mount a filesystem

`touch` : change file time

`umount` : un-mount a filesystem

# Two relevant xkcd comics




http://www.xkcd.org/376


http://www.xkcd.org/607

End of lecture