User accounts in UNIX

ComS 252 — Iowa State University

Andrew Miner

Motivation

This lecture gives an in depth look at UNIX–style user accounts

- Applies to Linux and most variants of UNIX
- Mac OS is a little different
  - E.g., different set of utilities
- But why in depth?

Motivation

This lecture gives an in depth look at UNIX–style user accounts

- Applies to Linux and most variants of UNIX
- Mac OS is a little different
    - E.g., different set of utilities
- But why in depth?
    1. You will need it for homework

Motivation

This lecture gives an in depth look at UNIX–style user accounts

- Applies to Linux and most variants of UNIX
- Mac OS is a little different
  - E.g., different set of utilities
- But why in depth?
  1. You will need it for homework
  2. You should know it anyway
     - Explains some potentially confusing behaviors
     - Helps keep your system secure

Motivation

This lecture gives an in depth look at UNIX–style user accounts

- Applies to Linux and most variants of UNIX
- Mac OS is a little different
  - E.g., different set of utilities
- But why in depth?
  1. You will need it for homework
  2. You should know it anyway
     - Explains some potentially confusing behaviors
     - Helps keep your system secure
  3. I have lied to you (as usual)
     - "Simplified things" is probably a nicer way to put it
     - Finally . . . the shocking truth

Managing user accounts and groups

1. GUI–based tools
   - E.g., `system-config-users`
   - Are for *wimps*
   - I assume you can figure these out
2. Command–line tools
   - As the demi–Gods intended
   - We will discuss these next. . .
3. By hand
   - Effective and safe if done correctly

Adding users

`useradd`

- Create a new user account
- Many switches, e.g. to specify
  - Location of user's home directory
  - User's primary (i.e., "default") group
  - Other groups the user belongs to
  - Default login shell
- Check your `man` pages for details

Pro tips for lazy system administrators:

- Lots of defaults may be specified in `/etc/login.defs`
- Can use a "skeleton" directory
  - User's home directory is copied from the skeleton

Adding users

`useradd`

- Create a new user account
- Many switches, e.g. to specify
  - Location of user's home directory
  - User's primary (i.e., "default") group
  - Other groups the user belongs to
  - Default login shell
- Check your `man` pages for details
- Only `root` may do this

Pro tips for lazy system administrators:

- Lots of defaults may be specified in `/etc/login.defs`
- Can use a "skeleton" directory
  - User's home directory is copied from the skeleton

Changing a user's account

usermod

- Change "anything" about a user's account
  - Even the username
- Switches for each thing to change
- Switches typically match those found in `useradd` (with some extras)
- Check your `man` pages for details

Changing a user's account

`usermod`

- Change "anything" about a user's account
  - Even the username
- Switches for each thing to change
- Switches typically match those found in `useradd` (with some extras)
- Check your `man` pages for details
- Only `root` may do this

Removing users

`userdel`

- Remove a user account

`-r` option:

  - Remove user's home directory and files
  - Other files must be tracked down "by hand"

- Check your `man` pages for details

Pro tip for lazy system administrators:

- Can specify an executable, in `/etc/login.defs`
- Will run with username as argument, whenever `userdel` is run
- Shell scripts are commonly used here
  - E.g., one that removes all files owned by that user (`find` can do this)

Removing users

userdel

- Remove a user account
- -r option:
  - Remove user's home directory and files
  - Other files must be tracked down "by hand"
- Check your `man` pages for details
- Only `root` may do this

Pro tip for lazy system administrators:

- Can specify an executable, in `/etc/login.defs`
- Will run with username as argument, whenever `userdel` is run
- Shell scripts are commonly used here
  - E.g., one that removes all files owned by that user (`find` can do this)

Groups

- There are group equivalents of the previous 3 utilities
- `groupadd`: create a new group
- `groupmod`: modify an existing group
- `groupdel`: remove an existing group
  - You cannot remove a user's primary group
- These tend to have only a few switches
- Check your `man` pages for details

Groups

- There are group equivalents of the previous 3 utilities
- `groupadd`: create a new group
- `groupmod`: modify an existing group
- `groupdel`: remove an existing group
  - You cannot remove a user's primary group
- These tend to have only a few switches
- Check your `man` pages for details
- Only `root` may run these

Changing your account — yourself

What can ordinary users do?

Changing your account — yourself

What can ordinary users do?

`chsh`

- Change your login shell
- Typically: must select one listed in `/etc/shells`

Changing your account — yourself

What can ordinary users do?

`chsh`

- Change your login shell
- Typically: must select one listed in `/etc/shells`

`passwd`

- Change your password
- You will be prompted for your current password
- You type a new password
  - Twice
- These days: checks the strength of the password

Running `passwd` as `root`

`passwd`

- Change the `root` password
- Works the same as for ordinary users
  - Prompt for current password
  - Type the new password, twice

Running `passwd` as `root`

`passwd`

- Change the `root` password
- Works the same as for ordinary users
  - Prompt for current password
  - Type the new password, twice

`passwd user`

- Change the password for `user`
- DOES NOT prompt for current password
- Type the new password, twice

Forgotten passwords

Normal user (in this case, “luser”)

1. Find a system administrator
2. She/he will run `passwd username` for you

Forgotten passwords

Normal user (in this case, “luser”)

1. Find a system administrator
2. She/he will run `passwd username` for you

`root`

Forgotten passwords

Normal user (in this case, “luser”)

1. Find a system administrator
2. She/he will run `passwd username` for you

`root`

1. Boot in single–user mode (runlevel 1)
   OR using “`rescue.target`” on `systemd`
2. Run `passwd`
3. Reboot

Forgotten passwords

Normal user (in this case, “luser”)

1. Find a system administrator
2. She/he will run `passwd username` for you

`root`

1. Boot in single–user mode (runlevel 1)
   OR using “`rescue.target`” on `systemd`
2. Run `passwd`
3. Reboot

How can I boot in single–user mode without `root` access?

Forgotten passwords

Normal user (in this case, "luser")

1. Find a system administrator
2. She/he will run `passwd username` for you

`root`

1. Boot in single–user mode (runlevel 1)
   OR using "`rescue.target`" on `systemd`
2. Run `passwd`
3. Reboot

How can I boot in single–user mode without `root` access?

- Can be done with `GrUB` at boot time, OR
- Boot from install DVD/CD in rescue mode

Current group

- When a user creates a new file, what group does it have?

Current group

- When a user creates a new file, what group does it have?
  - Whatever the “current group” is set to
    - Default upon login: the user’s primary group

Current group

- When a user creates a new file, what group does it have?
  - Whatever the “current group” is set to
    - Default upon login: the user’s primary group
- Can the current group be changed?

Current group

- When a user creates a new file, what group does it have?
  - Whatever the "current group" is set to
    - Default upon login: the user's primary group
- Can the current group be changed? Of course

`newgrp [groupname]`

- Change your current group to the one specified
  - If none specified — change to the primary group
- Will start a subshell
  - Exit this to get back the the previous group
  - Same idea as `su`
- What groups can I change to?

Current group

- When a user creates a new file, what group does it have?
  - Whatever the "current group" is set to
    - Default upon login: the user's primary group
- Can the current group be changed? Of course

`newgrp [groupname]`

- Change your current group to the one specified
  - If none specified — change to the primary group
- Will start a subshell
  - Exit this to get back the the previous group
  - Same idea as `su`
- What groups can I change to?
  - Whatever groups I belong to, right?

Current group

- When a user creates a new file, what group does it have?
  - Whatever the "current group" is set to
    - Default upon login: the user's primary group
- Can the current group be changed? Of course

`newgrp [groupname]`

- Change your current group to the one specified
  - If none specified — change to the primary group
- Will start a subshell
  - Exit this to get back the the previous group
  - Same idea as `su`
- What groups can I change to?
  - Whatever groups I belong to, right?
  - Normally, yes, but there is more . . .

Group administration

gpasswd

- Administer groups
- Lots of switches — check your `man` pages
- When run as `root`:
  - Can add and remove users
  - Can assign group administrators

Group administration

gpasswd

- Administer groups
- Lots of switches — check your `man` pages
- When run as `root`:
  - Can add and remove users
  - Can assign group administrators
  - Can set or remove a group password
- When run as a group administrator:
  - Can change the group password
  - Can add and remove users (I think . . .)
- Fun fact:
  - A group administrator does not need to belong to the group

Group passwords?

`newgrp`: the full story

When you try to change groups:

- If you are a member of the group
  - You do not need to type the group password
- If you are not a member of the group
  - If there is a group password
    - You are prompted for the group password
  - If there is not a group password: denied

Group passwords?

`newgrp`: the full story

When you try to change groups:

- If you are a member of the group
  - You do not need to type the group password
- If you are not a member of the group
  - If there is a group password
    - You are prompted for the group password
  - If there is not a group password: denied

I have seen conflicting documentation on this

Does anyone use group passwords on a production system?

- Probably not
  - If so I would like to know about it

Does anyone use group passwords on a production system?

- Probably not
    - If so I would like to know about it
- Why not?

Does anyone use group passwords on a production system?

- Probably not
  - If so I would like to know about it
- Why not?
- Shared passwords are much less secure
  - Difficult to enforce "don't share with everyone"
  - *Three can keep a secret, if two of them are dead*

— Benjamin Franklin

Does anyone use group passwords on a production system?

- Probably not
  - If so I would like to know about it
- Why not?
- Shared passwords are much less secure
  - Difficult to enforce “don’t share with everyone”
  - *Three can keep a secret, if two of them are dead*

    — Benjamin Franklin
- Logistics
  - Think about changing a group password

Does anyone use group passwords on a production system?

- Probably not
  - If so I would like to know about it
- Why not?
- Shared passwords are much less secure
  - Difficult to enforce "don't share with everyone"
  - *Three can keep a secret, if two of them are dead*

    — Benjamin Franklin
- Logistics
  - Think about changing a group password
- Adding or removing users has the same effect
  - Give me any scenario that uses group passwords
  - I will be able to either:
    1. Convince you that this violates security policy; or
    2. Find a way to use `gpasswd` to fix it, without a group password

What is a user ID?

What is a user ID?

User Identifier

- Kernel–level mechanism to identify users
- Is an unsigned integer
    - Range is at least 0 to 32,767

What is a user ID?

User Identifier

- Kernel–level mechanism to identify users
- Is an unsigned integer
  - Range is at least 0 to 32,767

- Superuser (`root`) always has user ID 0
  - Note: `root` is the username
- Low values are set aside for system use
  - E.g., 1 through 499 (RHEL) or 999 (Debian)

What is a user ID?

User Identifier

- Kernel–level mechanism to identify users
- Is an unsigned integer
  - Range is at least 0 to 32,767

- Superuser (`root`) always has user ID 0
  - Note: `root` is the username
- Low values are set aside for system use
  - E.g., 1 through 499 (RHEL) or 999 (Debian)

Group Identifier

- Kernel–level mechanism to identify groups
- Also an unsigned integer

How to find my user ID?

id

- Usage: `id [option]... [username]`
- Print user identity for specified user (default: current user)
- Also prints the current group (or primary group)
- Check your `man` pages for options

```
prompt$ █
```

How to find my user ID?

id

- Usage: `id [option]... [username]`
- Print user identity for specified user (default: current user)
- Also prints the current group (or primary group)
- Check your `man` pages for options

```
prompt$ id
```

How to find my user ID?

id

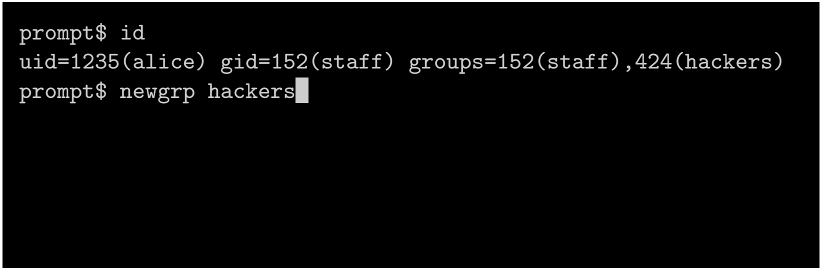
- Usage: `id [option]... [username]`
- Print user identity for specified user (default: current user)
- Also prints the current group (or primary group)
- Check your `man` pages for options

```
prompt$ id
uid=1235(alice) gid=152(staff) groups=152(staff),424(hackers)
prompt$
```

How to find my user ID?

id

- Usage: `id [option]... [username]`
- Print user identity for specified user (default: current user)
- Also prints the current group (or primary group)
- Check your `man` pages for options



```
prompt$ id
uid=1235(alice) gid=152(staff) groups=152(staff),424(hackers)
prompt$ newgrp hackers
```

How to find my user ID?

`id`

- Usage: `id [option]... [username]`
- Print user identity for specified user (default: current user)
- Also prints the current group (or primary group)
- Check your `man` pages for options

```
prompt$ id
uid=1235(alice) gid=152(staff) groups=152(staff),424(hackers)
prompt$ newgrp hackers
prompt$ █
```

How to find my user ID?

id

- Usage: `id [option]... [username]`
- Print user identity for specified user (default: current user)
- Also prints the current group (or primary group)
- Check your `man` pages for options

```
prompt$ id
uid=1235(alice) gid=152(staff) groups=152(staff),424(hackers)
prompt$ newgrp hackers
prompt$ id
```

How to find my user ID?

id

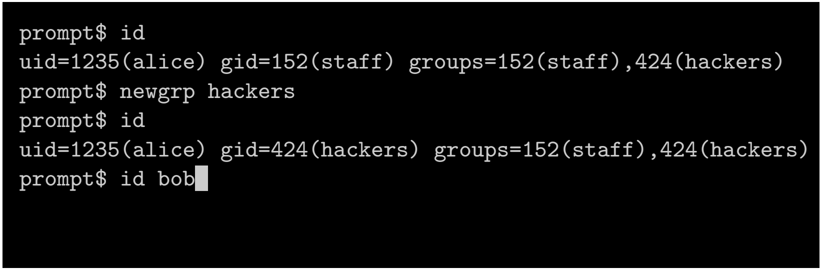
- Usage: `id [option]... [username]`
- Print user identity for specified user (default: current user)
- Also prints the current group (or primary group)
- Check your `man` pages for options

```
prompt$ id
uid=1235(alice) gid=152(staff) groups=152(staff),424(hackers)
prompt$ newgrp hackers
prompt$ id
uid=1235(alice) gid=424(hackers) groups=152(staff),424(hackers)
prompt$ 
```

How to find my user ID?

id

- Usage: `id [option]... [username]`
- Print user identity for specified user (default: current user)
- Also prints the current group (or primary group)
- Check your `man` pages for options



```
prompt$ id
uid=1235(alice) gid=152(staff) groups=152(staff),424(hackers)
prompt$ newgrp hackers
prompt$ id
uid=1235(alice) gid=424(hackers) groups=152(staff),424(hackers)
prompt$ id bob
```

How to find my user ID?

id

- Usage: `id [option]... [username]`
- Print user identity for specified user (default: current user)
- Also prints the current group (or primary group)
- Check your `man` pages for options

```
prompt$ id
uid=1235(alice) gid=152(staff) groups=152(staff),424(hackers)
prompt$ newgrp hackers
prompt$ id
uid=1235(alice) gid=424(hackers) groups=152(staff),424(hackers)
prompt$ id bob
uid=1239(bob) gid=152(staff) groups=152(staff),207(webadmin)
prompt$ █
```

Why this is important

Why this is important

- How does the ext*n* filesystem store the owner and group for each file and directory?

Why this is important

- How does the `ext`*n* filesystem store the owner and group for each file and directory?
  - The file's owner is stored as the userid
  - The file's group is stored as the groupid

Why this is important

- How does the ext*n* filesystem store the owner and group for each file and directory?
  - The file's owner is stored as the userid
  - The file's group is stored as the groupid
- You can see this using the `-n` switch for `ls`

Why this is important

- How does the `ext`*n* filesystem store the owner and group for each file and directory?
  - The file's owner is stored as the userid
  - The file's group is stored as the groupid
- You can see this using the `-n` switch for `ls`
- And you need to care about this because. . .

Why this is important

- How does the ext*n* filesystem store the owner and group for each file and directory?
  - The file's owner is stored as the userid
  - The file's group is stored as the groupid
- You can see this using the `-n` switch for `ls`
- And you need to care about this because. . .
- What happens if you mount a disk on another machine?

Why this is important

- How does the `ext`*n* filesystem store the owner and group for each file and directory?
  - The file's owner is stored as the userid
  - The file's group is stored as the groupid
- You can see this using the `-n` switch for `ls`
- And you need to care about this because. . .
- What happens if you mount a disk on another machine?
  - The file owners and groups may change
  - Happens if the userids and groupids do not "match" on the two machines

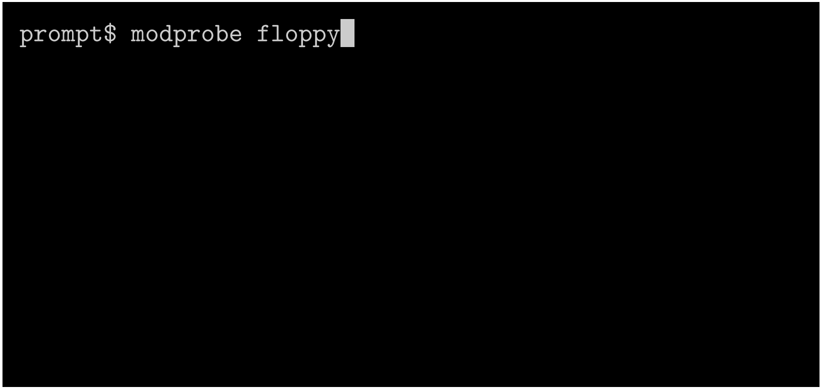
Concrete example: part 1

Machine A: copy stuff to a floppy (as `root`)

```
prompt$ █
```

Concrete example: part 1

Machine A: copy stuff to a floppy (as `root`)



```
prompt$ modprobe floppy
```

Concrete example: part 1

Machine A: copy stuff to a floppy (as `root`)

```
prompt$ modprobe floppy
prompt$ █
```

Concrete example: part 1

Machine A: copy stuff to a floppy (as `root`)

```
prompt$ modprobe floppy
prompt$ mkfs -t ext2 /dev/fd0
```

Concrete example: part 1

Machine A: copy stuff to a floppy (as `root`)

```
prompt$ modprobe floppy
prompt$ mkfs -t ext2 /dev/fd0
prompt$ █
```

Concrete example: part 1

Machine A: copy stuff to a floppy (as `root`)

```
prompt$ modprobe floppy
prompt$ mkfs -t ext2 /dev/fd0
prompt$ mkdir /mnt/floppy
```

Concrete example: part 1

Machine A: copy stuff to a floppy (as `root`)

```
prompt$ modprobe floppy
prompt$ mkfs -t ext2 /dev/fd0
prompt$ mkdir /mnt/floppy
prompt$
```

Concrete example: part 1

Machine A: copy stuff to a floppy (as `root`)

```
prompt$ modprobe floppy
prompt$ mkfs -t ext2 /dev/fd0
prompt$ mkdir /mnt/floppy
prompt$ mount -t ext2 /dev/fd0 /mnt/floppy
```

Concrete example: part 1

Machine A: copy stuff to a floppy (as `root`)

```
prompt$ modprobe floppy
prompt$ mkfs -t ext2 /dev/fd0
prompt$ mkdir /mnt/floppy
prompt$ mount -t ext2 /dev/fd0 /mnt/floppy
prompt$ █
```

Concrete example: part 1

Machine A: copy stuff to a floppy (as `root`)

```
prompt$ modprobe floppy
prompt$ mkfs -t ext2 /dev/fd0
prompt$ mkdir /mnt/floppy
prompt$ mount -t ext2 /dev/fd0 /mnt/floppy
prompt$ cp -p /tmp/* /mnt/floppy
```

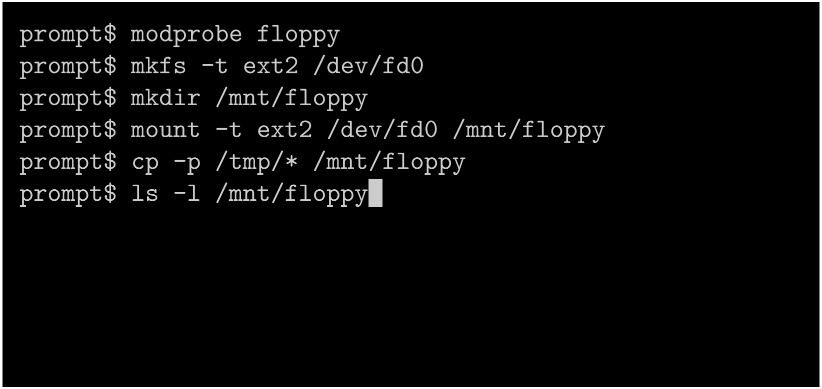
Concrete example: part 1

Machine A: copy stuff to a floppy (as `root`)

```
prompt$ modprobe floppy
prompt$ mkfs -t ext2 /dev/fd0
prompt$ mkdir /mnt/floppy
prompt$ mount -t ext2 /dev/fd0 /mnt/floppy
prompt$ cp -p /tmp/* /mnt/floppy
prompt$ 
```

Concrete example: part 1

Machine A: copy stuff to a floppy (as `root`)



```
prompt$ modprobe floppy
prompt$ mkfs -t ext2 /dev/fd0
prompt$ mkdir /mnt/floppy
prompt$ mount -t ext2 /dev/fd0 /mnt/floppy
prompt$ cp -p /tmp/* /mnt/floppy
prompt$ ls -l /mnt/floppy
```

Concrete example: part 1

Machine A: copy stuff to a floppy (as `root`)

```
prompt$ modprobe floppy
prompt$ mkfs -t ext2 /dev/fd0
prompt$ mkdir /mnt/floppy
prompt$ mount -t ext2 /dev/fd0 /mnt/floppy
prompt$ cp -p /tmp/* /mnt/floppy
prompt$ ls -l /mnt/floppy
-rw------- 1 alice  staff   1024 Dec 25  2010 card.txt
-rw------- 1 bob    bob     8001 Apr 22 23:41 fooB6WGlB
-rw-r--r-- 1 chuck  web     3239 Jul  5 13:02 hello.html
drwx------ 2 root   root   12288 Nov  1 12:45 lost+found
prompt$ █
```

Concrete example: part 1

Machine A: copy stuff to a floppy (as `root`)

```
prompt$ modprobe floppy
prompt$ mkfs -t ext2 /dev/fd0
prompt$ mkdir /mnt/floppy
prompt$ mount -t ext2 /dev/fd0 /mnt/floppy
prompt$ cp -p /tmp/* /mnt/floppy
prompt$ ls -l /mnt/floppy
-rw------- 1 alice  staff   1024 Dec 25  2010 card.txt
-rw------- 1 bob    bob     8001 Apr 22 23:41 fooB6WG1B
-rw-r--r-- 1 chuck  web     3239 Jul  5 13:02 hello.html
drwx------ 2 root   root   12288 Nov  1 12:45 lost+found
prompt$ ls -ln /mnt/floppy
```

Concrete example: part 1

Machine A: copy stuff to a floppy (as `root`)

```
prompt$ ls -l /mnt/floppy
-rw------- 1 alice  staff   1024 Dec 25  2010 card.txt
-rw------- 1 bob    bob     8001 Apr 22 23:41 fooB6WG1B
-rw-r--r-- 1 chuck  web     3239 Jul  5 13:02 hello.html
drwx------ 2 root   root   12288 Nov  1 12:45 lost+found
prompt$ ls -ln /mnt/floppy
-rw------- 1 1001  401   1024 Dec 25  2010 card.txt
-rw------- 1 1002  513   8001 Apr 22 23:41 fooB6WG1B
-rw-r--r-- 1 1003  425   3239 Jul 05 13:02 hello.html
drwx------ 2 0     0    12288 Nov  1 12:45 lost+found
prompt$ █
```

Concrete example: part 1

Machine A: copy stuff to a floppy (as `root`)

```
prompt$ ls -l /mnt/floppy
-rw------- 1 alice  staff   1024 Dec 25  2010 card.txt
-rw------- 1 bob    bob     8001 Apr 22 23:41 fooB6WGlB
-rw-r--r-- 1 chuck  web     3239 Jul  5 13:02 hello.html
drwx------ 2 root   root   12288 Nov  1 12:45 lost+found
prompt$ ls -ln /mnt/floppy
-rw------- 1 1001  401    1024 Dec 25  2010 card.txt
-rw------- 1 1002  513    8001 Apr 22 23:41 fooB6WGlB
-rw-r--r-- 1 1003  425    3239 Jul 05 13:02 hello.html
drwx------ 2 0     0     12288 Nov  1 12:45 lost+found
prompt$ umount /mnt/floppy
```

Concrete example: part 1

Machine A: copy stuff to a floppy (as `root`)

```
-rw------- 1 alice   staff    1024 Dec 25  2010 card.txt
-rw------- 1 bob     bob      8001 Apr 22 23:41 fooB6WGlB
-rw-r--r-- 1 chuck   web      3239 Jul  5 13:02 hello.html
drwx------ 2 root    root    12288 Nov  1 12:45 lost+found
prompt$ ls -ln /mnt/floppy
-rw------- 1 1001  401   1024 Dec 25  2010 card.txt
-rw------- 1 1002  513   8001 Apr 22 23:41 fooB6WGlB
-rw-r--r-- 1 1003  425   3239 Jul 05 13:02 hello.html
drwx------ 2 0     0    12288 Nov  1 12:45 lost+found
prompt$ umount /mnt/floppy
prompt$ 
```

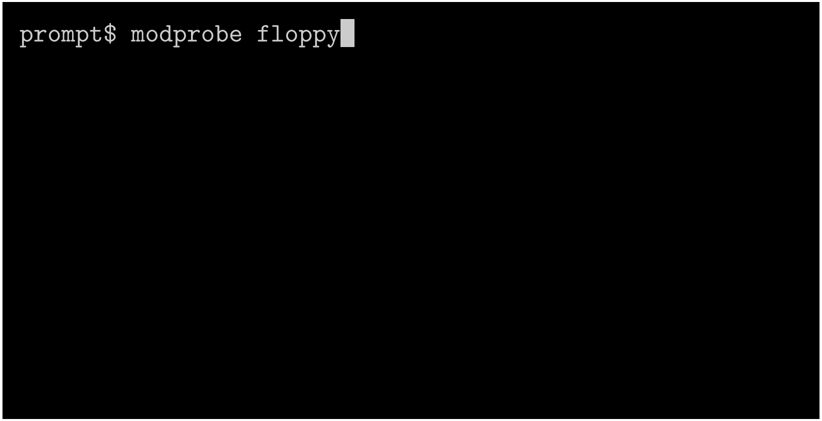
Concrete example: part 2

Machine B: copy stuff off the floppy (as `root`)

```
prompt$ █
```

Concrete example: part 2

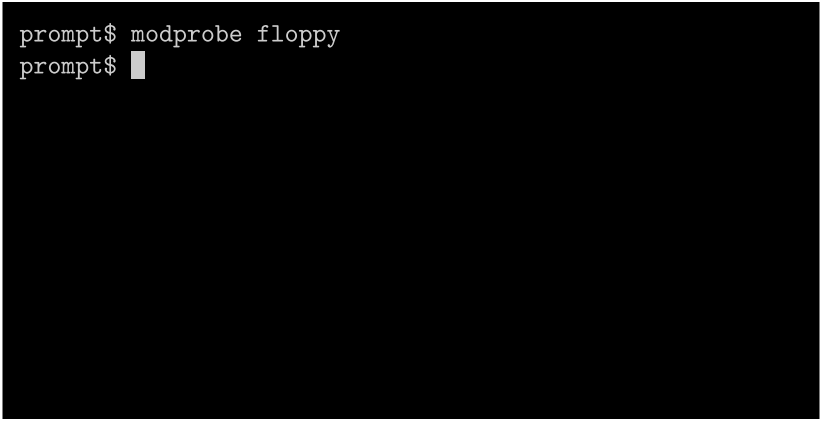
Machine B: copy stuff off the floppy (as `root`)



```
prompt$ modprobe floppy
```

Concrete example: part 2

Machine B: copy stuff off the floppy (as `root`)



```
prompt$ modprobe floppy
prompt$ 
```

Concrete example: part 2

Machine B: copy stuff off the floppy (as `root`)

```
prompt$ modprobe floppy
prompt$ mkdir /mnt/floppy
```

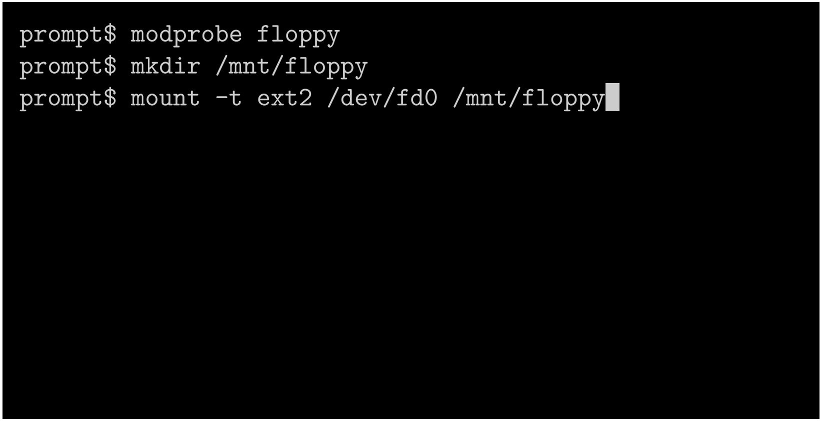
Concrete example: part 2

Machine B: copy stuff off the floppy (as `root`)

```
prompt$ modprobe floppy
prompt$ mkdir /mnt/floppy
prompt$ █
```

Concrete example: part 2

Machine B: copy stuff off the floppy (as `root`)



```
prompt$ modprobe floppy
prompt$ mkdir /mnt/floppy
prompt$ mount -t ext2 /dev/fd0 /mnt/floppy
```

Concrete example: part 2

Machine B: copy stuff off the floppy (as `root`)

```
prompt$ modprobe floppy
prompt$ mkdir /mnt/floppy
prompt$ mount -t ext2 /dev/fd0 /mnt/floppy
prompt$ █
```

Concrete example: part 2

Machine B: copy stuff off the floppy (as `root`)

```
prompt$ modprobe floppy
prompt$ mkdir /mnt/floppy
prompt$ mount -t ext2 /dev/fd0 /mnt/floppy
prompt$ ls -l /mnt/floppy
```

Concrete example: part 2

Machine B: copy stuff off the floppy (as `root`)

```
prompt$ modprobe floppy
prompt$ mkdir /mnt/floppy
prompt$ mount -t ext2 /dev/fd0 /mnt/floppy
prompt$ ls -l /mnt/floppy
-rw------- 1 chuck  users   1024 Dec 25  2010 card.txt
-rw------- 1 dave    513    8001 Apr 22 23:41 fooB6WGlB
-rw-r--r-- 1 frank  noobs   3239 Jul  5 13:02 hello.html
drwx------ 2 root   root   12288 Nov  1 12:45 lost+found
prompt$ █
```

Concrete example: part 2

Machine B: copy stuff off the floppy (as `root`)

```
prompt$ modprobe floppy
prompt$ mkdir /mnt/floppy
prompt$ mount -t ext2 /dev/fd0 /mnt/floppy
prompt$ ls -l /mnt/floppy
-rw------- 1 chuck  users   1024 Dec 25  2010 card.txt
-rw------- 1 dave    513    8001 Apr 22 23:41 fooB6WGlB
-rw-r--r-- 1 frank  noobs   3239 Jul  5 13:02 hello.html
drwx------ 2 root   root   12288 Nov  1 12:45 lost+found
prompt$ ls -ln /mnt/floppy
```

Concrete example: part 2

Machine B: copy stuff off the floppy (as `root`)

```
prompt$ mount -t ext2 /dev/fd0 /mnt/floppy
prompt$ ls -l /mnt/floppy
-rw------- 1 chuck  users   1024 Dec 25  2010 card.txt
-rw------- 1 dave    513    8001 Apr 22 23:41 fooB6WGlB
-rw-r--r-- 1 frank  noobs   3239 Jul  5 13:02 hello.html
drwx------ 2 root   root   12288 Nov  1 12:45 lost+found
prompt$ ls -ln /mnt/floppy
-rw------- 1 1001 401   1024 Dec 25  2010 card.txt
-rw------- 1 1002 513   8001 Apr 22 23:41 fooB6WGlB
-rw-r--r-- 1 1003 425   3239 Jul 05 13:02 hello.html
drwx------ 2 0     0    12288 Nov  1 12:45 lost+found
prompt$ █
```

Concrete example: part 2

Machine B: copy stuff off the floppy (as `root`)

```
prompt$ mount -t ext2 /dev/fd0 /mnt/floppy
prompt$ ls -l /mnt/floppy
-rw------- 1 chuck  users   1024 Dec 25  2010 card.txt
-rw------- 1 dave    513    8001 Apr 22 23:41 fooB6WG1B
-rw-r--r-- 1 frank  noobs   3239 Jul  5 13:02 hello.html
drwx------ 2 root   root   12288 Nov  1 12:45 lost+found
prompt$ ls -ln /mnt/floppy
-rw------- 1 1001  401   1024 Dec 25  2010 card.txt
-rw------- 1 1002  513   8001 Apr 22 23:41 fooB6WG1B
-rw-r--r-- 1 1003  425   3239 Jul 05 13:02 hello.html
drwx------ 2 0     0    12288 Nov  1 12:45 lost+found
prompt$ umount /mnt/floppy
```

Concrete example: part 2

Machine B: copy stuff off the floppy (as `root`)

```
prompt$ ls -l /mnt/floppy
-rw------- 1 chuck  users   1024 Dec 25  2010 card.txt
-rw------- 1 dave    513    8001 Apr 22 23:41 fooB6WGlB
-rw-r--r-- 1 frank  noobs   3239 Jul  5 13:02 hello.html
drwx------ 2 root   root   12288 Nov  1 12:45 lost+found
prompt$ ls -ln /mnt/floppy
-rw------- 1 1001  401   1024 Dec 25  2010 card.txt
-rw------- 1 1002  513   8001 Apr 22 23:41 fooB6WGlB
-rw-r--r-- 1 1003  425   3239 Jul 05 13:02 hello.html
drwx------ 2 0     0    12288 Nov  1 12:45 lost+found
prompt$ umount /mnt/floppy
prompt$ █
```

Concrete example: part 2

Machine B: copy stuff off the floppy (as `root`)

```
prompt$ ls -l /mnt/floppy
-rw------- 1 chuck  users   1024 Dec 25  2010 card.txt
-rw------- 1 dave    513    8001 Apr 22 23:41 fooB6WGlB
-rw-r--r-- 1 frank  noobs   3239 Jul  5 13:02 hello.html
drwx------ 2 root   root   12288 Nov  1 12:45 lost+found
prompt$ ls -ln /mnt/floppy
-rw------- 1 1001  401   1024 Dec 25  2010 card.txt
-rw------- 1 1002  513   8001 Apr 22 23:41 fooB6WGlB
-rw-r--r-- 1 1003  425   3239 Jul 05 13:02 hello.html
drwx------ 2 0     0    12288 Nov  1 12:45 lost+found
prompt$ umount /mnt/floppy
prompt$ id frank
```

Concrete example: part 2

Machine B: copy stuff off the floppy (as `root`)

```
-rw------- 1 dave    513    8001 Apr 22 23:41 fooB6WGlB
-rw-r--r-- 1 frank  noobs   3239 Jul  5 13:02 hello.html
drwx------ 2 root   root   12288 Nov  1 12:45 lost+found
prompt$ ls -ln /mnt/floppy
-rw------- 1 1001  401   1024 Dec 25  2010 card.txt
-rw------- 1 1002  513   8001 Apr 22 23:41 fooB6WGlB
-rw-r--r-- 1 1003  425   3239 Jul 05 13:02 hello.html
drwx------ 2 0     0    12288 Nov  1 12:45 lost+found
prompt$ umount /mnt/floppy
prompt$ id frank
uid=1003(frank) gid=401(users) groups=401(users)
prompt$ █
```

Changing users and groups "by hand"

- Done in the usual way:
  - Find the appropriate file(s)
  - Change the appropriate "magic text"
    - But need to do this properly
- Let's see where the files are, and how they are formatted

Wait a minute

- We will discuss the inner workings of UNIX passwords?

Wait a minute

- We will discuss the inner workings of UNIX passwords?
  - Yes

Wait a minute

- We will discuss the inner workings of UNIX passwords?
  - Yes
- But . . . what about system security and stuff?

Wait a minute

- We will discuss the inner workings of UNIX passwords?
    - Yes
- But . . . what about system security and stuff?
    1. This material is common knowledge already
        - And not everyone that knows it is "nice"
    2. Relying on "security through obscurity" is generally a bad idea

Wait a minute

- We will discuss the inner workings of UNIX passwords?
    - Yes
- But . . . what about system security and stuff?
    1. This material is common knowledge already
        - And not everyone that knows it is “nice”
    2. Relying on “security through obscurity” is generally a bad idea
- But isn’t all security done by obscuring something?

Wait a minute

- We will discuss the inner workings of UNIX passwords?
  - Yes
- But . . . what about system security and stuff?
  1. This material is common knowledge already
     - And not everyone that knows it is "nice"
  2. Relying on "security through obscurity" is generally a bad idea
- But isn't all security done by obscuring something?
  - "Security through obscurity" keeps the inner workings secret
  - But there are publicly–known algorithms that provide security
  - Compare this to an algorithm that relies on being secret
  - We can discuss this more when we get to "Security"

/etc/passwd

- World–readable text file
- One user "record" per line
- Each record has these fields, separated by ":"

/etc/passwd

- World–readable text file
- One user “record” per line
- Each record has these fields, separated by “:”
  1. Username

/etc/passwd

- World–readable text file
- One user "record" per line
- Each record has these fields, separated by ":"
  1. Username
  2. Password
     - Ancient systems: password information was here
     - Modern systems: "x" or "*" here

/etc/passwd

- World–readable text file
- One user "record" per line
- Each record has these fields, separated by ":"
  1. Username
  2. Password
     - Ancient systems: password information was here
     - Modern systems: "x" or "*" here
  3. UserID

/etc/passwd

- World–readable text file
- One user "record" per line
- Each record has these fields, separated by ":"
    1. Username
    2. Password
        - Ancient systems: password information was here
        - Modern systems: "x" or "*" here
    3. UserID
    4. Primary groupID

/etc/passwd

- World–readable text file
- One user "record" per line
- Each record has these fields, separated by ":"
  1. Username
  2. Password
     - Ancient systems: password information was here
     - Modern systems: "x" or "*" here
  3. UserID
  4. Primary groupID
  5. GECOS
     - Usually, user's full name

/etc/passwd

- World–readable text file
- One user “record” per line
- Each record has these fields, separated by “:”
  1. Username
  2. Password
     - Ancient systems: password information was here
     - Modern systems: “x” or “*” here
  3. UserID
  4. Primary groupID
  5. GECOS
     - Usually, user’s full name
  6. Home directory

/etc/passwd

- World–readable text file
- One user "record" per line
- Each record has these fields, separated by ":"
  1. Username
  2. Password
     - Ancient systems: password information was here
     - Modern systems: "x" or "*" here
  3. UserID
  4. Primary groupID
  5. GECOS
     - Usually, user's full name
  6. Home directory
  7. Default shell

/etc/passwd

- World–readable text file
- One user "record" per line
- Each record has these fields, separated by ":"
  1. Username
  2. Password
     - Ancient systems: password information was here
     - Modern systems: "x" or "*" here
  3. UserID
  4. Primary groupID
  5. GECOS
     - Usually, user's full name
  6. Home directory
  7. Default shell

```
prompt$ █
```

/etc/passwd

- World–readable text file
- One user “record” per line
- Each record has these fields, separated by “:”
  1. Username
  2. Password
     - Ancient systems: password information was here
     - Modern systems: “x” or “*” here
  3. UserID
  4. Primary groupID
  5. GECOS
     - Usually, user’s full name
  6. Home directory
  7. Default shell

```
prompt$ tail -n 1 /etc/passwd
```

/etc/passwd

- World–readable text file
- One user “record” per line
- Each record has these fields, separated by “:”
  1. Username
  2. Password
     - Ancient systems: password information was here
     - Modern systems: “x” or “*” here
  3. UserID
  4. Primary groupID
  5. GECOS
     - Usually, user’s full name
  6. Home directory
  7. Default shell

```
prompt$ tail -n 1 /etc/passwd
chuck:x:502:502:Carlos R. Norris:/home/chuck:/bin/bash
prompt$ █
```

/etc/shadow

- NOT world–readable
  - Permissions tend to be 000
- One user “record” per line
- Each record has these fields, separated by “:”

/etc/shadow

- NOT world–readable
  - Permissions tend to be 000
- One user "record" per line
- Each record has these fields, separated by ":"
  1. Username

/etc/shadow

- NOT world–readable
  - Permissions tend to be 000
- One user “record” per line
- Each record has these fields, separated by “:”
  1. Username
  2. Password (“encrypted”)
     - Or “!!” to indicate no password

/etc/shadow

- NOT world–readable
  - Permissions tend to be 000
- One user “record” per line
- Each record has these fields, separated by “:”
  1. Username
  2. Password (“encrypted”)
     - Or “!!” to indicate no password
  3. Date of last password change
     - Format is “number of days since Jan 1, 1970”

/etc/shadow

- NOT world–readable
  - Permissions tend to be 000
- One user “record” per line
- Each record has these fields, separated by “:”
  1. Username
  2. Password (“encrypted”)
     - Or “!!” to indicate no password
  3. Date of last password change
     - Format is “number of days since Jan 1, 1970”
  4. How many days until a password change is allowed

/etc/shadow

- NOT world–readable
  - Permissions tend to be 000
- One user "record" per line
- Each record has these fields, separated by ":"
  1. Username
  2. Password ("encrypted")
     - Or "!!" to indicate no password
  3. Date of last password change
     - Format is "number of days since Jan 1, 1970"
  4. How many days until a password change is allowed
  5. How many days until password expires

/etc/shadow

- NOT world–readable
  - Permissions tend to be 000
- One user “record” per line
- Each record has these fields, separated by “:”
  1. Username
  2. Password (“encrypted”)
     - Or “!!” to indicate no password
  3. Date of last password change
     - Format is “number of days since Jan 1, 1970”
  4. How many days until a password change is allowed
  5. How many days until password expires
  6. Number of days before expiration that warnings are given

/etc/shadow

- NOT world–readable
  - Permissions tend to be 000
- One user "record" per line
- Each record has these fields, separated by ":"
  1. Username
  2. Password ("encrypted")
     - Or "!!" to indicate no password
  3. Date of last password change
     - Format is "number of days since Jan 1, 1970"
  4. How many days until a password change is allowed
  5. How many days until password expires
  6. Number of days before expiration that warnings are given
  7. Number of days after expiration that account is disabled

/etc/shadow

- NOT world–readable
  - Permissions tend to be 000
- One user "record" per line
- Each record has these fields, separated by ":"
  1. Username
  2. Password ("encrypted")
     - Or "!!" to indicate no password
  3. Date of last password change
     - Format is "number of days since Jan 1, 1970"
  4. How many days until a password change is allowed
  5. How many days until password expires
  6. Number of days before expiration that warnings are given
  7. Number of days after expiration that account is disabled
  8. Date when the account is disabled
     - Format is "number of days since Jan 1, 1970"

How is the password encrypted?

How is the password encrypted?

Using a cryptographic hash function

How is the password encrypted?

Using a cryptographic hash function

- Takes a string as input
  - In this case, the password plus a “salt” string
  - “salt” is random text, and not secret
- Produces a random–looking string as output
  - “salt” is added to the output string

How is the password encrypted?

Using a cryptographic hash function

- Takes a string as input
  - In this case, the password plus a "salt" string
  - "salt" is random text, and not secret
- Produces a random–looking string as output
  - "salt" is added to the output string

Properties of an ideal cyrptographic hash function

1. Easy: compute the output string, for any input string
2. Infeasible: find an input string for a given output string
3. Infeasible: change input string without changing output string
4. Infeasible: find two different input strings with same output

Infeasible: possible, but time required is too long to be useful

How does password authentication work?

1. User types password
2. System pulls salt and hashed password from `/etc/shadow`
3. Typed password and salt are plugged into hash function
4. If output matches hashed password then
   - We are “sure” that the passwords match
     - But could be extremely low probability of no match
     - I.e., we found another input string with the same output
5. Otherwise we know the passwords did not match

/etc/group

- World–readable text file
- One group “record” per line
- Each record has these fields, separated by “:”

/etc/group

- World–readable text file
- One group "record" per line
- Each record has these fields, separated by ":"
    1. Group name

/etc/group

- World–readable text file
- One group “record” per line
- Each record has these fields, separated by “`:`”
  1. Group name
  2. Password
     - Ancient systems: password information was here
     - Modern systems: “`x`” or “`*`” here

/etc/group

- World–readable text file
- One group “record” per line
- Each record has these fields, separated by “:”
  1. Group name
  2. Password
     - Ancient systems: password information was here
     - Modern systems: “x” or “*” here
  3. GroupID

/etc/group

- World–readable text file
- One group “record” per line
- Each record has these fields, separated by “:”
  1. Group name
  2. Password
     - Ancient systems: password information was here
     - Modern systems: “x” or “*” here
  3. GroupID
  4. Comma separated list of group members (usernames)

/etc/group

- World–readable text file
- One group “record” per line
- Each record has these fields, separated by “:”
  1. Group name
  2. Password
     - Ancient systems: password information was here
     - Modern systems: “x” or “*” here
  3. GroupID
  4. Comma separated list of group members (usernames)

```
prompt$ █
```

/etc/group

- World–readable text file
- One group "record" per line
- Each record has these fields, separated by ":"
  1. Group name
  2. Password
     - Ancient systems: password information was here
     - Modern systems: "x" or "*" here
  3. GroupID
  4. Comma separated list of group members (usernames)

```
prompt$ tail -n 1 /etc/group
```

/etc/group

- World–readable text file
- One group "record" per line
- Each record has these fields, separated by ":"
    1. Group name
    2. Password
        - Ancient systems: password information was here
        - Modern systems: "x" or "*" here
    3. GroupID
    4. Comma separated list of group members (usernames)

```
prompt$ tail -n 1 /etc/group
hackers:x:600:alice,bob
prompt$ █
```

/etc/gshadow

- Not world–readable text file (usually has permissions 000)
- One group "record" per line
- Each record has these fields, separated by ":"

/etc/gshadow

- Not world–readable text file (usually has permissions 000)
- One group “record” per line
- Each record has these fields, separated by “:”
    1. Group name

/etc/gshadow

- Not world–readable text file (usually has permissions 000)
- One group "record" per line
- Each record has these fields, separated by ":"
  1. Group name
  2. Password (encrypted)
     - Or "!" to indicate no password

/etc/gshadow

- Not world–readable text file (usually has permissions 000)
- One group “record” per line
- Each record has these fields, separated by “:”
  1. Group name
  2. Password (encrypted)
     - Or “!” to indicate no password
  3. Comma separated list of group administrators (usernames)

/etc/gshadow

- Not world–readable text file (usually has permissions 000)
- One group "record" per line
- Each record has these fields, separated by ":"
    1. Group name
    2. Password (encrypted)
        - Or "!" to indicate no password
    3. Comma separated list of group administrators (usernames)
    4. Comma separated list of group members (usernames)

/etc/gshadow

- Not world–readable text file (usually has permissions 000)
- One group "record" per line
- Each record has these fields, separated by ":"
  1. Group name
  2. Password (encrypted)
     - Or "!" to indicate no password
  3. Comma separated list of group administrators (usernames)
  4. Comma separated list of group members (usernames)

```
prompt$ █
```

/etc/gshadow

- Not world–readable text file (usually has permissions 000)
- One group "record" per line
- Each record has these fields, separated by ":"
  1. Group name
  2. Password (encrypted)
     - Or "!" to indicate no password
  3. Comma separated list of group administrators (usernames)
  4. Comma separated list of group members (usernames)

```
prompt$ tail -n 1 /etc/gshadow
```

/etc/gshadow

- Not world–readable text file (usually has permissions 000)
- One group "record" per line
- Each record has these fields, separated by ":"
  1. Group name
  2. Password (encrypted)
     - Or "!" to indicate no password
  3. Comma separated list of group administrators (usernames)
  4. Comma separated list of group members (usernames)

```
prompt$ tail -n 1 /etc/gshadow
hackers:!::alice,bob
prompt$ █
```

Changing things

- Do the earlier utilities
  - `useradd`, `passwd`, `gpasswd`, etc.,

  just modify the files
  - `/etc/passwd`, `/etc/shadow`,
  - `/etc/group`, `/etc/gshadow`

  as appropriate?

Changing things

- Do the earlier utilities
  - `useradd`, `passwd`, `gpasswd`, etc.,

  just modify the files
  - `/etc/passwd`, `/etc/shadow`,
  - `/etc/group`, `/etc/gshadow`

  as appropriate? Yes.

Changing things

- Do the earlier utilities
  - `useradd`, `passwd`, `gpasswd`, etc.,

  just modify the files
  - `/etc/passwd`, `/etc/shadow`,
  - `/etc/group`, `/etc/gshadow`

  as appropriate? Yes.

- Can I edit those files myself, by hand?

Changing things

- Do the earlier utilities
  - `useradd`, `passwd`, `gpasswd`, etc.,

  just modify the files
  - `/etc/passwd`, `/etc/shadow`,
  - `/etc/group`, `/etc/gshadow`

  as appropriate? Yes.

- Can I edit those files myself, by hand? Yes.

Changing things

- Do the earlier utilities
  - `useradd`, `passwd`, `gpasswd`, etc.,

  just modify the files
  - `/etc/passwd`, `/etc/shadow`,
  - `/etc/group`, `/etc/gshadow`

  as appropriate? Yes.

- Can I edit those files myself, by hand? Yes.
- Just using `vi`, `emacs`, `nano`, or whatever?

Changing things

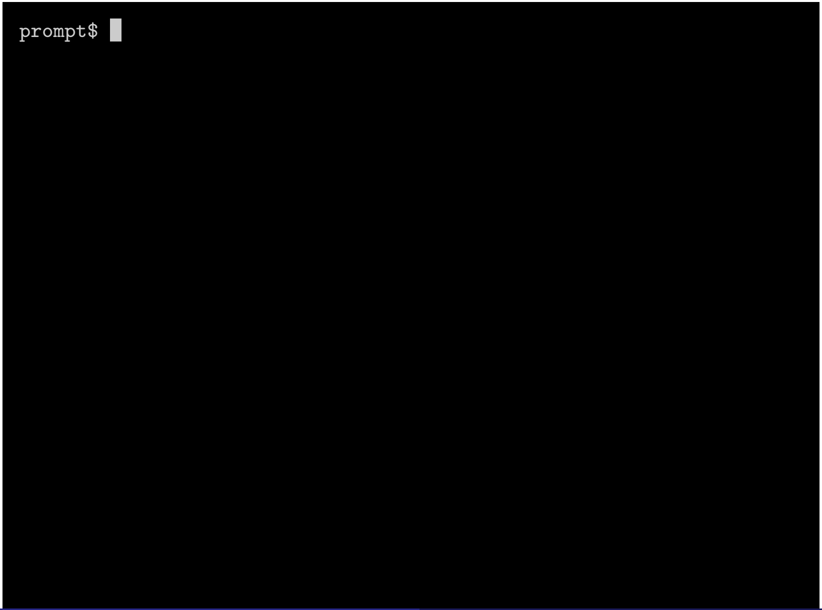
- Do the earlier utilities
  - `useradd`, `passwd`, `gpasswd`, etc.,

  just modify the files
  - `/etc/passwd`, `/etc/shadow`,
  - `/etc/group`, `/etc/gshadow`

  as appropriate? Yes.

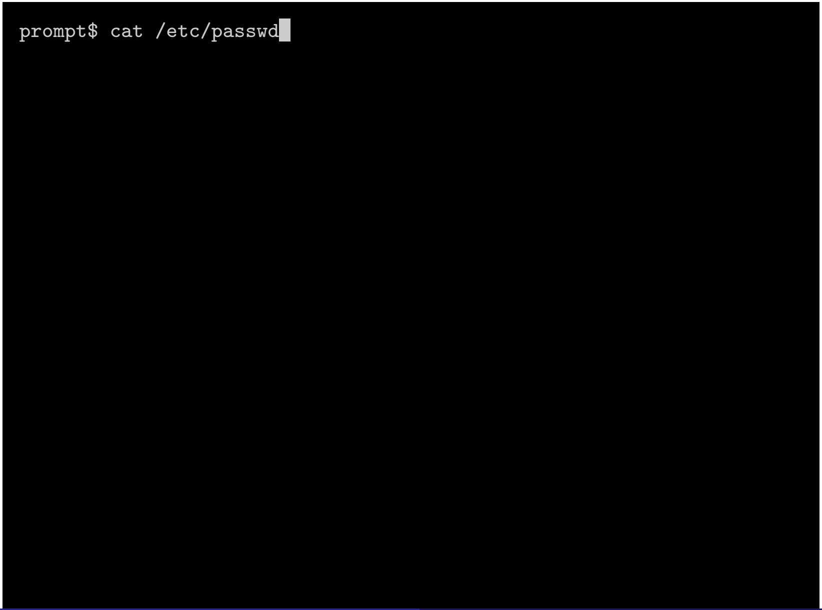
- Can I edit those files myself, by hand? Yes.
- Just using `vi`, `emacs`, `nano`, or whatever?
  - NO — dangerous
    - Files are not locked, could be corrupted:
    - Someone could run a utility while you are editing
  - Safe way:
    - `vipw /etc/passwd` or `vipw -s /etc/shadow`
    - `vigr /etc/group` or `vigr -s /etc/gshadow`
    - Locks files to prevent corruption

Let's have a look at an actual `/etc/passwd` from one of the VMs:



```
prompt$ 
```

Let's have a look at an actual `/etc/passwd` from one of the VMs:



```
prompt$ cat /etc/passwd
```

Let's have a look at an actual `/etc/passwd` from one of the VMs:

```
prompt$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
saslauth:x:499:499:"Saslauthd user":/var/empty/saslauth:/sbin/nologin
mailnull:x:47:47::/var/spool/mqueue:/sbin/nologin
smmsp:x:51:51::/var/spool/mqueue:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
user:x:500:500::/home/user:/bin/bash
prompt$ █
```

Why so many system accounts?

- Usually for running daemons
- Why not just run these as `root`?

Why so many system accounts?

- Usually for running daemons
- Why not just run these as `root`?
  - General security rule: don't do it as `root` unless necessary

Why so many system accounts?

- Usually for running daemons
- Why not just run these as `root`?
  - General security rule: don't do it as `root` unless necessary
- But only `root` processes can `bind` to reserved ports!

Why so many system accounts?

- Usually for running daemons
- Why not just run these as `root`?
  - General security rule: don't do it as `root` unless necessary
- But only `root` processes can `bind` to reserved ports!
  - True.
  - But, we can:

Why so many system accounts?

- Usually for running daemons
- Why not just run these as `root`?
  - General security rule: don't do it as `root` unless necessary
- But only `root` processes can `bind` to reserved ports!
  - True.
  - But, we can:
    1. Start out running as `root`

Why so many system accounts?

- Usually for running daemons
- Why not just run these as `root`?
  - General security rule: don't do it as `root` unless necessary
- But only `root` processes can `bind` to reserved ports!
  - True.
  - But, we can:
    1. Start out running as `root`
    2. Get our ports set up

Why so many system accounts?

- Usually for running daemons
- Why not just run these as `root`?
  - General security rule: don't do it as `root` unless necessary
- But only `root` processes can `bind` to reserved ports!
  - True.
  - But, we can:
    1. Start out running as `root`
    2. Get our ports set up
    3. Have the process EITHER:
       switch to an ordinary user, OR
       create a new process running as an ordinary user

Fun aside

What is the likely purpose of this account in `/etc/passwd`?

```
lpd:x:57:57:lpd:/var/spool/lpd:/bin/bash
```

Fun aside

What is the likely purpose of this account in `/etc/passwd`?

```
lpd:x:57:57:lpd:/var/spool/lpd:/bin/bash
```

- The system account for the printing daemon?

Fun aside

What is the likely purpose of this account in `/etc/passwd`?

```
lpd:x:57:57:lpd:/var/spool/lpd:/bin/bash
```

- The system account for the printing daemon?
  - NO
  - That's `lp`

Fun aside

What is the likely purpose of this account in `/etc/passwd`?

```
    lpd:x:57:57:lpd:/var/spool/lpd:/bin/bash
```

- The system account for the printing daemon?
  - NO
  - That's `lp`
- Some random system account that should be left alone?

Fun aside

What is the likely purpose of this account in `/etc/passwd`?

```
lpd:x:57:57:lpd:/var/spool/lpd:/bin/bash
```

- The system account for the printing daemon?
  - NO
  - That's `lp`
- Some random system account that should be left alone? NO

Fun aside

What is the likely purpose of this account in `/etc/passwd`?

```
lpd:x:57:57:lpd:/var/spool/lpd:/bin/bash
```

- The system account for the printing daemon?
  - NO
  - That's `lp`
- Some random system account that should be left alone? NO
- A cracker who has broken into the system?

Fun aside

What is the likely purpose of this account in `/etc/passwd`?

```
lpd:x:57:57:lpd:/var/spool/lpd:/bin/bash
```

- The system account for the printing daemon?
  - NO
  - That's `lp`
- Some random system account that should be left alone? NO
- A cracker who has broken into the system?
  - YES!

Fun aside

What is the likely purpose of this account in `/etc/passwd`?

```
lpd:x:57:57:lpd:/var/spool/lpd:/bin/bash
```

- The system account for the printing daemon?
  - NO
  - That's `lp`
- Some random system account that should be left alone? NO
- A cracker who has broken into the system?
  - YES!
  - System accounts should have shell: `/sbin/nologin`

Fun aside

What is the likely purpose of this account in `/etc/passwd`?

```
lpd:x:57:57:lpd:/var/spool/lpd:/bin/bash
```

- The system account for the printing daemon?
  - NO
  - That's `lp`
- Some random system account that should be left alone? NO
- A cracker who has broken into the system?
  - YES!
  - System accounts should have shell: `/sbin/nologin`
- But "low userIDs are set aside for system use", aren't they?

Fun aside

What is the likely purpose of this account in `/etc/passwd`?

```
lpd:x:57:57:lpd:/var/spool/lpd:/bin/bash
```

- The system account for the printing daemon?
  - NO
  - That's `lp`
- Some random system account that should be left alone? NO
- A cracker who has broken into the system?
  - YES!
  - System accounts should have shell: `/sbin/nologin`
- But "low userIDs are set aside for system use", aren't they?
  - Just a convention, `root` can do anything
  - This person probably got `root` access and created the account

Fun aside

What is the likely purpose of this account in `/etc/passwd`?

```
lpd:x:57:57:lpd:/var/spool/lpd:/bin/bash
```

- The system account for the printing daemon?
  - NO
  - That's `lp`
- Some random system account that should be left alone? NO
- A cracker who has broken into the system?
  - YES!
  - System accounts should have shell: `/sbin/nologin`
- But "low userIDs are set aside for system use", aren't they?
  - Just a convention, `root` can do anything
  - This person probably got `root` access and created the account

Paranoid yet?

And now for a very serious question

And now for a very serious question

How is it possible for an ordinary user to change their own password?

And now for a very serious question

How is it possible for an ordinary user to change their own password?

Consider the following statements

1. The `passwd` utility modifies file `/etc/shadow` directly
2. `/etc/shadow` typically has permissions 000
3. Whenever a user runs a process,
   the process runs with that user's permissions
4. An ordinary user can change their own password using `passwd`

And now for a very serious question

How is it possible for an ordinary user to change their own password?

Consider the following statements

1. The `passwd` utility modifies file `/etc/shadow` directly
2. `/etc/shadow` typically has permissions 000
3. Whenever a user runs a process,
   the process runs with that user's permissions
4. An ordinary user can change their own password using `passwd`

These statements cannot all be true. Which one is incorrect?

And now for a very serious question

How is it possible for an ordinary user to change their own password?

Consider the following statements

1. The `passwd` utility modifies file `/etc/shadow` directly
2. `/etc/shadow` typically has permissions 000
3. Whenever a user runs a process, the process runs with that user's permissions
4. An ordinary user can change their own password using `passwd`

These statements cannot all be true. Which one is incorrect?

The truth about processes

The truth about processes

A process has multiple userIDs and groupIDs:

- real user ID
  - User who started the process; its "owner"
  - C system call: `getuid()` to obtain this
- real group ID
  - Current group of user who started the process
  - C system call: `getgid()` to obtain this
- effective user ID
  - User ID to use for file permissions
  - *Usually* the same as the real user ID
  - C system call: `geteuid()` to obtain this
- effective group ID
  - Group ID to use for file permissions
  - *Usually* the same as the group user ID
  - C system call: `getegid()` to obtain this

Why alice can change her password

When `alice` runs `passwd`:

- The real user ID is `alice`'s
- The effective user ID is `root`'s (0)

Why `alice` can change her password

When `alice` runs `passwd`:

- The real user ID is `alice`'s
- The effective user ID is `root`'s (0)

By what magic does this happen?

setuid and setgid bits

- There are two permission bits we have not discussed

setuid bit	:	set user ID upon execution
		▶ When set, the process's effective user ID is set to the owner of the executable file
setgid bit	:	set group ID upon execution
		▶ When set, the process's effective group ID is set to the group of the executable file

- These can be changed with `chmod`
  - Use "`s`" where you would use "`r`", "`w`", or "`x`"
  - Use an extra octal digit before the usual three

4	:	setuid bit is on
2	:	setgid bit is on
1	:	sticky bit is on

- These can be seen in `ls -l`
  - Will have an "`s`" instead of "`x`" in the appropriate place

setuid and setgid bits

- There are two permission bits we have not discussed

setuid bit	:	set user ID upon execution
		▶ When set, the process's effective user ID is set to the owner of the executable file
setgid bit	:	set group ID upon execution
		▶ When set, the process's effective group ID is set to the group of the executable file

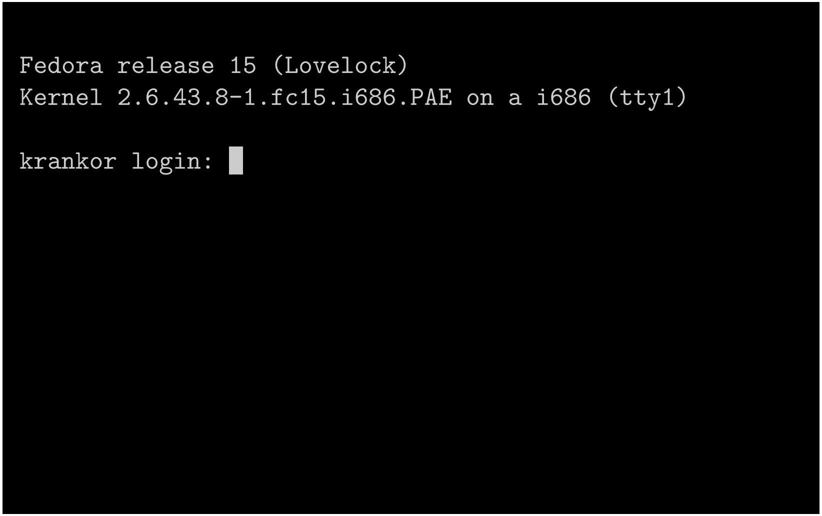
- These can be changed with `chmod`
  - Use "`s`" where you would use "`r`", "`w`", or "`x`"
  - Use an extra octal digit before the usual three

4	:	setuid bit is on
2	:	setgid bit is on
1	:	sticky bit is on

- These can be seen in `ls -l`
  - Will have an "`s`" instead of "`x`" in the appropriate place

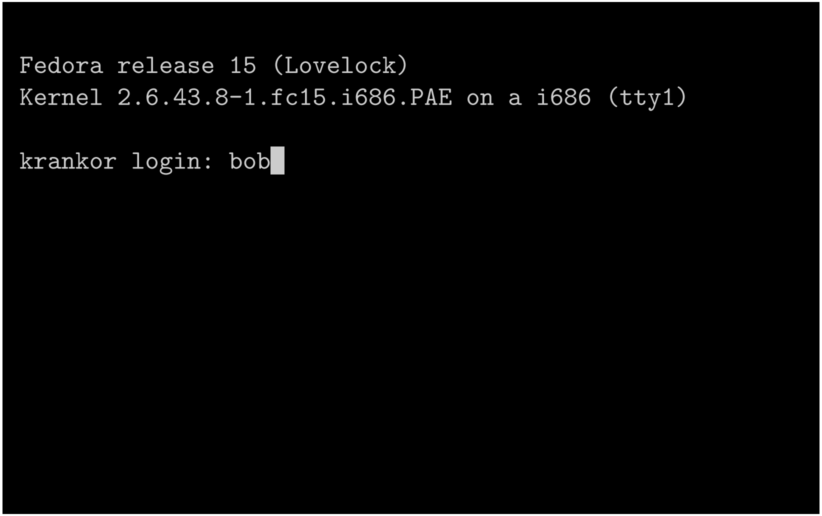
BE VERY CAREFUL WITH THESE BITS — we'll see why later

Fun setuid example



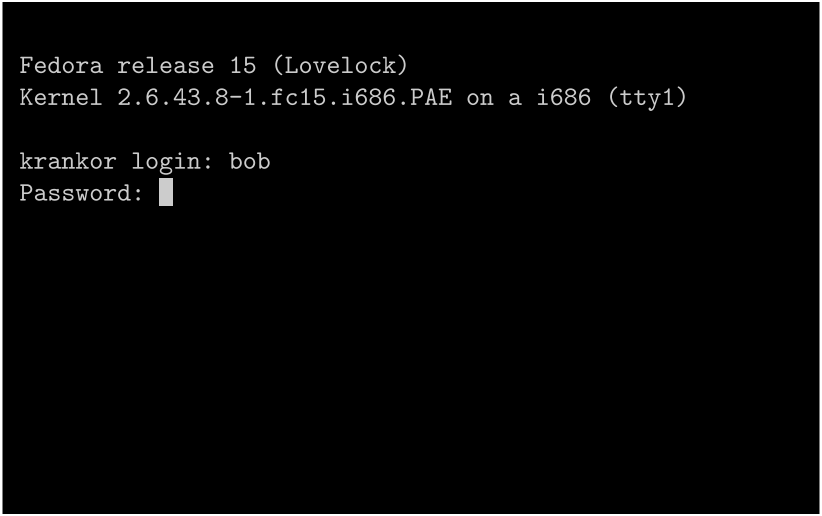
```
Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: █
```

Fun setuid example



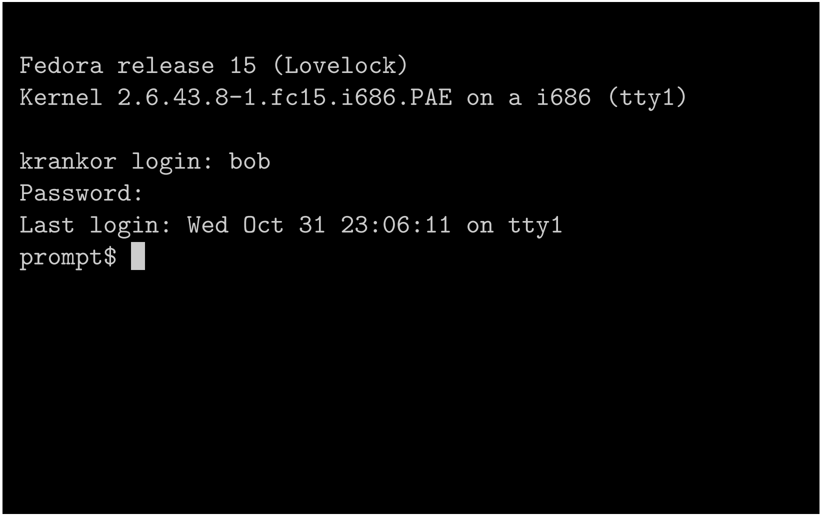
```
Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: bob
```

Fun setuid example



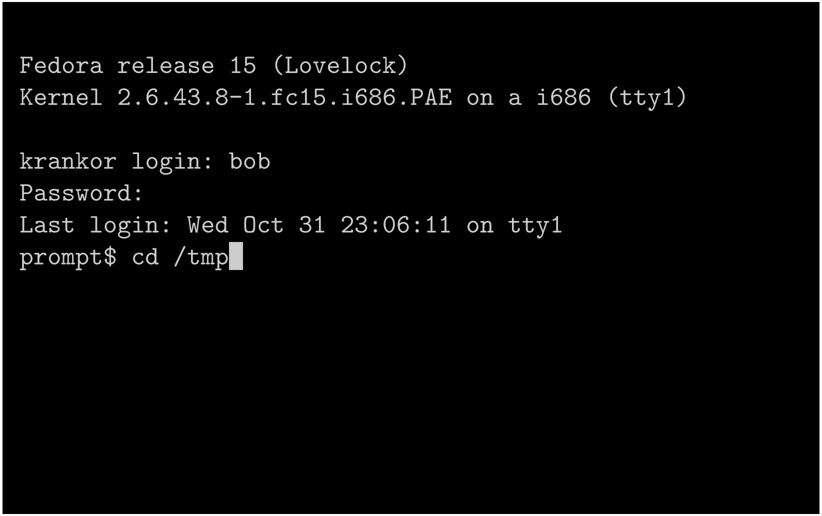
```
Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: bob
Password:
```

Fun setuid example



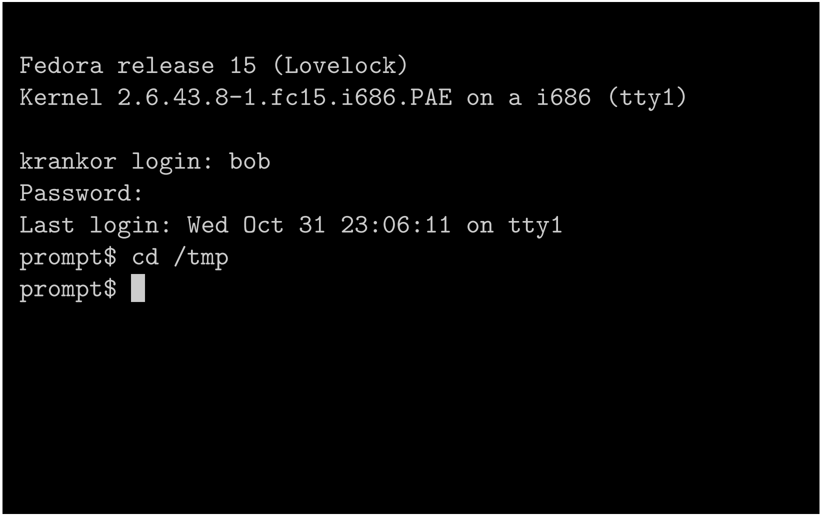
```
Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: bob
Password:
Last login: Wed Oct 31 23:06:11 on tty1
prompt$ █
```

Fun setuid example



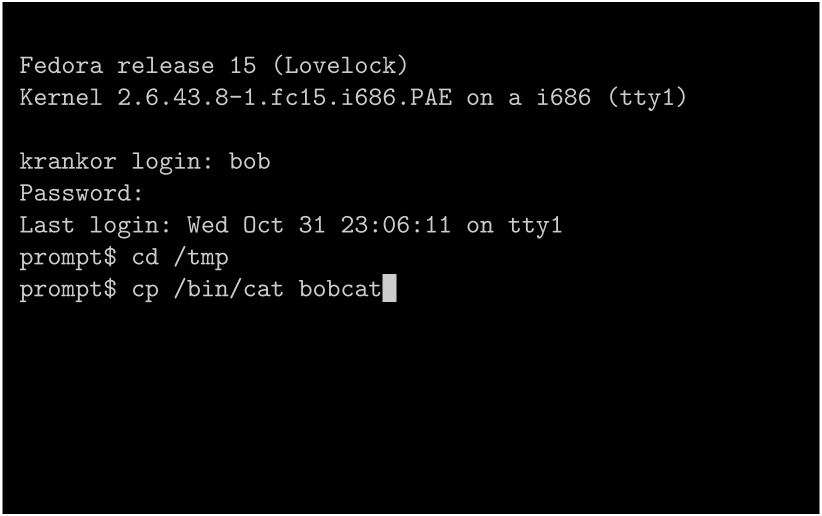
```
Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: bob
Password:
Last login: Wed Oct 31 23:06:11 on tty1
prompt$ cd /tmp
```

Fun setuid example



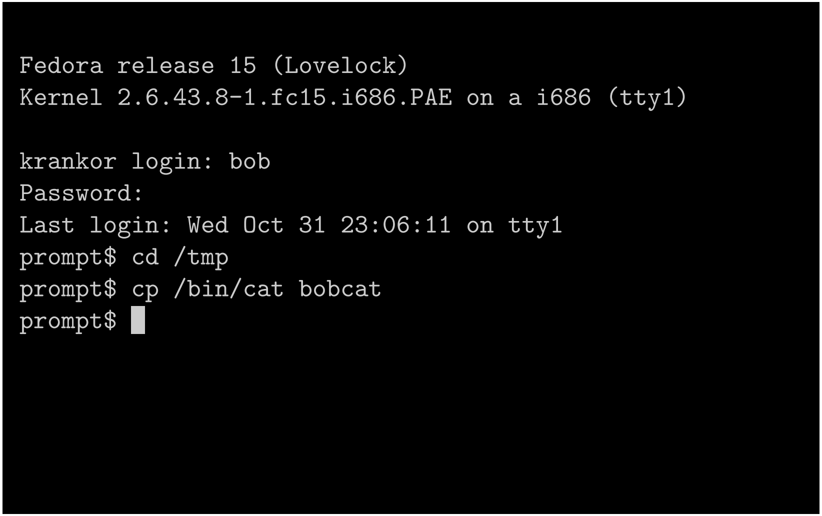
```
Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: bob
Password:
Last login: Wed Oct 31 23:06:11 on tty1
prompt$ cd /tmp
prompt$ █
```

Fun setuid example



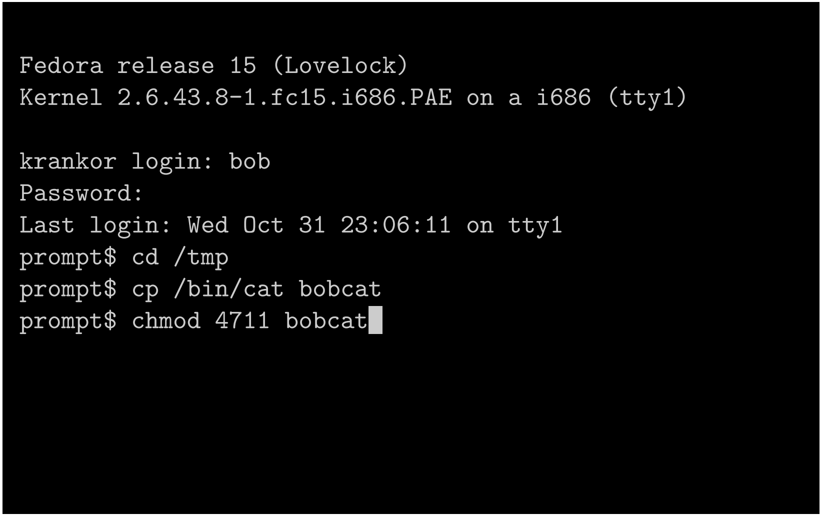
```
Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: bob
Password:
Last login: Wed Oct 31 23:06:11 on tty1
prompt$ cd /tmp
prompt$ cp /bin/cat bobcat
```

Fun setuid example



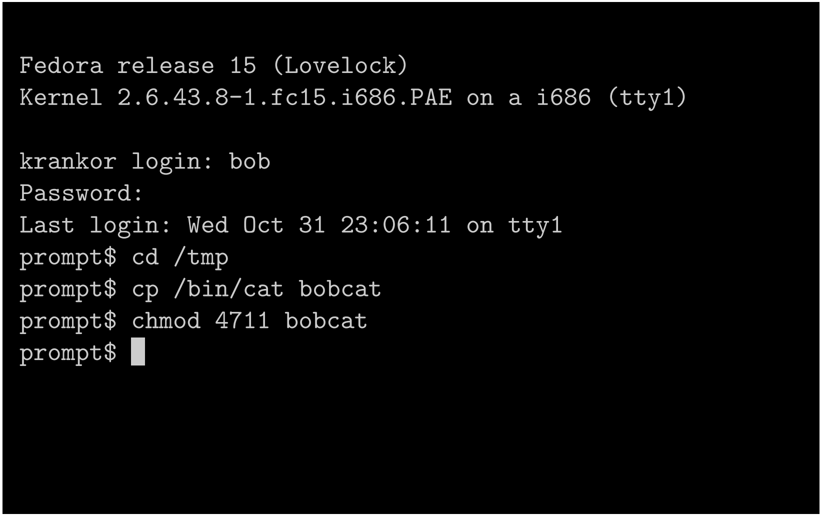
```
Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: bob
Password:
Last login: Wed Oct 31 23:06:11 on tty1
prompt$ cd /tmp
prompt$ cp /bin/cat bobcat
prompt$ █
```

Fun setuid example



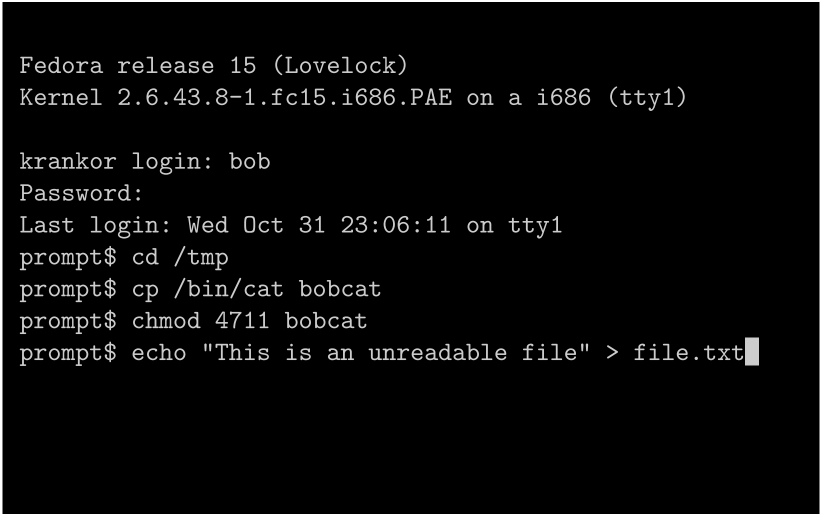
```
Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: bob
Password:
Last login: Wed Oct 31 23:06:11 on tty1
prompt$ cd /tmp
prompt$ cp /bin/cat bobcat
prompt$ chmod 4711 bobcat
```

Fun setuid example



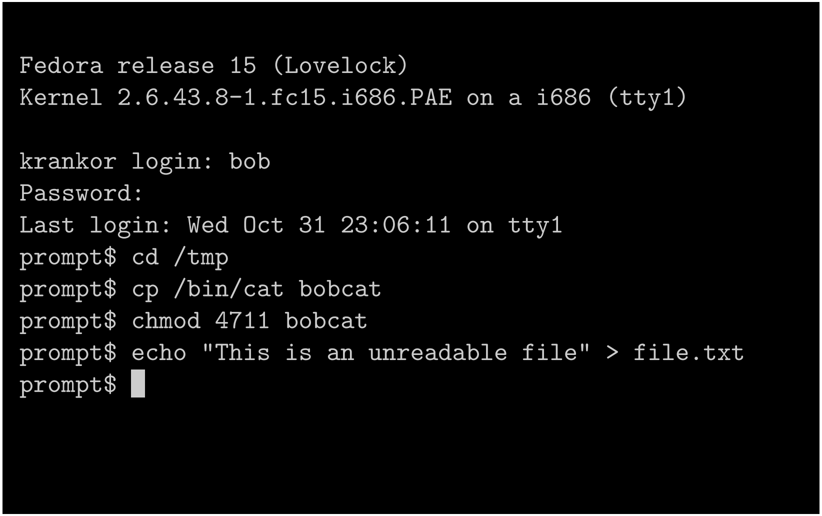
```
Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: bob
Password:
Last login: Wed Oct 31 23:06:11 on tty1
prompt$ cd /tmp
prompt$ cp /bin/cat bobcat
prompt$ chmod 4711 bobcat
prompt$ 
```

Fun setuid example



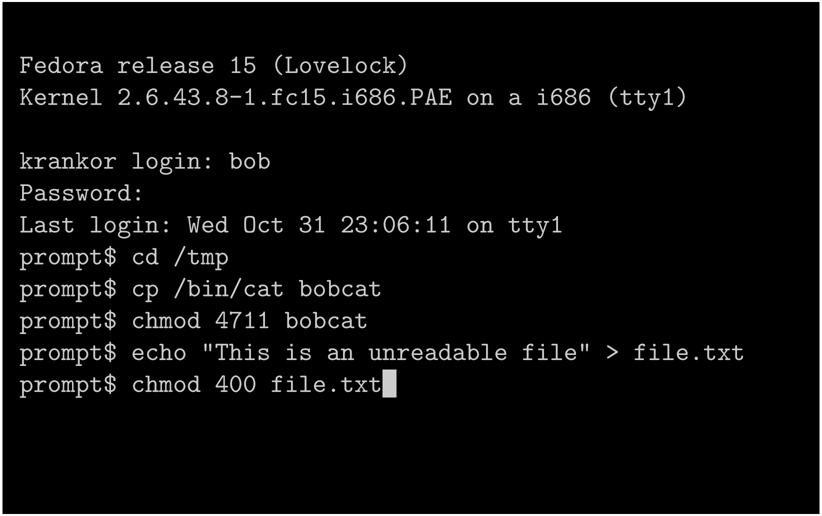
```
Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: bob
Password:
Last login: Wed Oct 31 23:06:11 on tty1
prompt$ cd /tmp
prompt$ cp /bin/cat bobcat
prompt$ chmod 4711 bobcat
prompt$ echo "This is an unreadable file" > file.txt
```

Fun setuid example



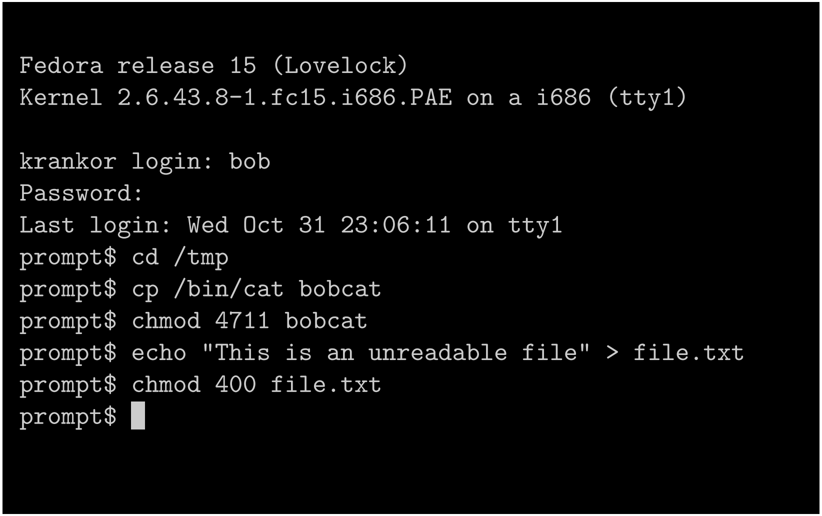
```
Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: bob
Password:
Last login: Wed Oct 31 23:06:11 on tty1
prompt$ cd /tmp
prompt$ cp /bin/cat bobcat
prompt$ chmod 4711 bobcat
prompt$ echo "This is an unreadable file" > file.txt
prompt$ 
```

Fun setuid example



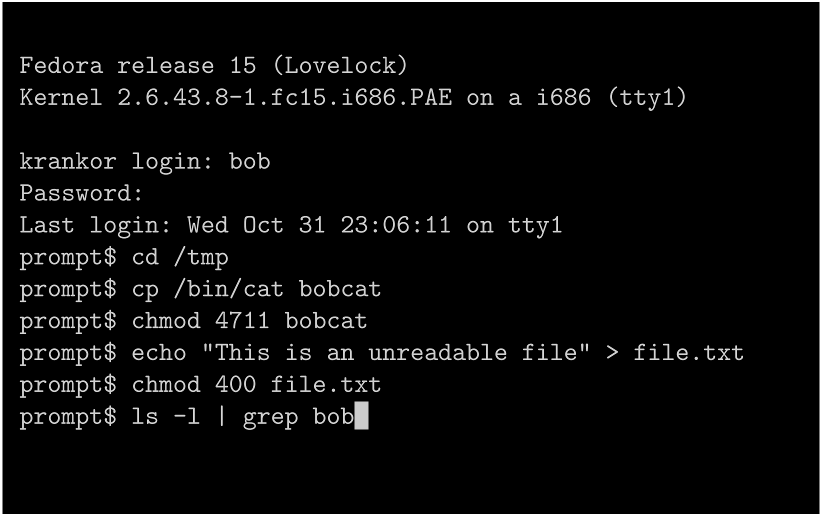
```
Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: bob
Password:
Last login: Wed Oct 31 23:06:11 on tty1
prompt$ cd /tmp
prompt$ cp /bin/cat bobcat
prompt$ chmod 4711 bobcat
prompt$ echo "This is an unreadable file" > file.txt
prompt$ chmod 400 file.txt
```

Fun setuid example



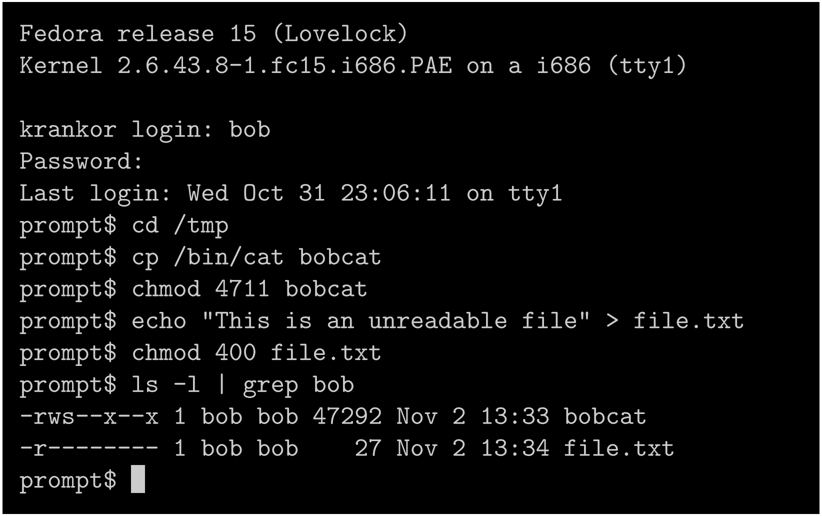
```
Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: bob
Password:
Last login: Wed Oct 31 23:06:11 on tty1
prompt$ cd /tmp
prompt$ cp /bin/cat bobcat
prompt$ chmod 4711 bobcat
prompt$ echo "This is an unreadable file" > file.txt
prompt$ chmod 400 file.txt
prompt$ █
```

Fun setuid example



```
Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: bob
Password:
Last login: Wed Oct 31 23:06:11 on tty1
prompt$ cd /tmp
prompt$ cp /bin/cat bobcat
prompt$ chmod 4711 bobcat
prompt$ echo "This is an unreadable file" > file.txt
prompt$ chmod 400 file.txt
prompt$ ls -l | grep bob
```

Fun setuid example

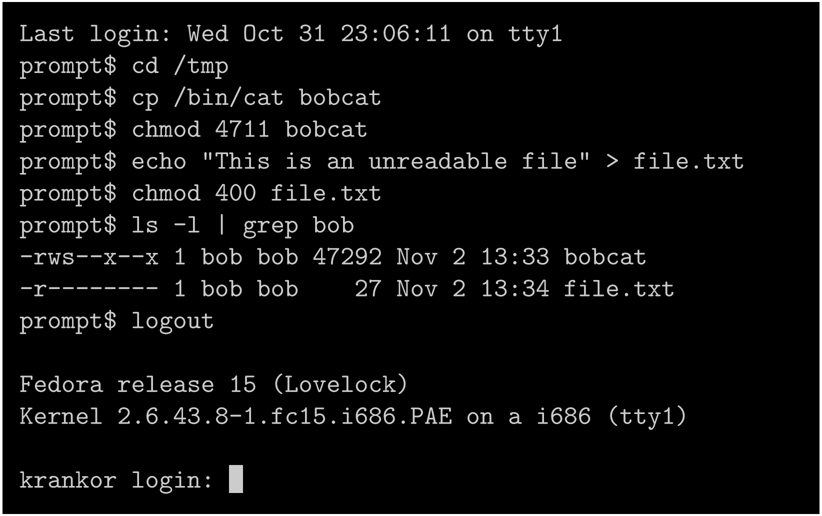


```
Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: bob
Password:
Last login: Wed Oct 31 23:06:11 on tty1
prompt$ cd /tmp
prompt$ cp /bin/cat bobcat
prompt$ chmod 4711 bobcat
prompt$ echo "This is an unreadable file" > file.txt
prompt$ chmod 400 file.txt
prompt$ ls -l | grep bob
-rws--x--x 1 bob bob 47292 Nov 2 13:33 bobcat
-r-------- 1 bob bob    27 Nov 2 13:34 file.txt
prompt$
```

Fun setuid example

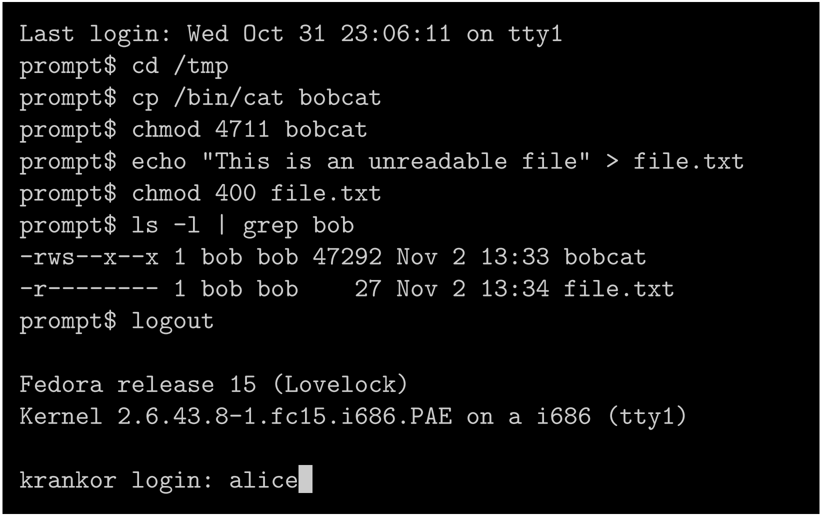
```
Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: bob
Password:
Last login: Wed Oct 31 23:06:11 on tty1
prompt$ cd /tmp
prompt$ cp /bin/cat bobcat
prompt$ chmod 4711 bobcat
prompt$ echo "This is an unreadable file" > file.txt
prompt$ chmod 400 file.txt
prompt$ ls -l | grep bob
-rws--x--x 1 bob bob 47292 Nov 2 13:33 bobcat
-r-------- 1 bob bob    27 Nov 2 13:34 file.txt
prompt$ logout
```

Fun setuid example



```
Last login: Wed Oct 31 23:06:11 on tty1
prompt$ cd /tmp
prompt$ cp /bin/cat bobcat
prompt$ chmod 4711 bobcat
prompt$ echo "This is an unreadable file" > file.txt
prompt$ chmod 400 file.txt
prompt$ ls -l | grep bob
-rws--x--x 1 bob bob 47292 Nov 2 13:33 bobcat
-r-------- 1 bob bob    27 Nov 2 13:34 file.txt
prompt$ logout

Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login:
```

Fun setuid example



```
Last login: Wed Oct 31 23:06:11 on tty1
prompt$ cd /tmp
prompt$ cp /bin/cat bobcat
prompt$ chmod 4711 bobcat
prompt$ echo "This is an unreadable file" > file.txt
prompt$ chmod 400 file.txt
prompt$ ls -l | grep bob
-rws--x--x 1 bob bob 47292 Nov 2 13:33 bobcat
-r-------- 1 bob bob    27 Nov 2 13:34 file.txt
prompt$ logout

Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: alice
```

Fun setuid example

```
prompt$ cd /tmp
prompt$ cp /bin/cat bobcat
prompt$ chmod 4711 bobcat
prompt$ echo "This is an unreadable file" > file.txt
prompt$ chmod 400 file.txt
prompt$ ls -l | grep bob
-rws--x--x 1 bob bob 47292 Nov 2 13:33 bobcat
-r-------- 1 bob bob    27 Nov 2 13:34 file.txt
prompt$ logout

Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: alice
Password: █
```

Fun setuid example

```
prompt$ chmod 4711 bobcat
prompt$ echo "This is an unreadable file" > file.txt
prompt$ chmod 400 file.txt
prompt$ ls -l | grep bob
-rws--x--x 1 bob bob 47292 Nov 2 13:33 bobcat
-r-------- 1 bob bob    27 Nov 2 13:34 file.txt
prompt$ logout

Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: alice
Password:
Last login: Thu Nov 1 17:12:23 on tty1
prompt$ █
```

Fun setuid example

```
prompt$ chmod 4711 bobcat
prompt$ echo "This is an unreadable file" > file.txt
prompt$ chmod 400 file.txt
prompt$ ls -l | grep bob
-rws--x--x 1 bob bob 47292 Nov 2 13:33 bobcat
-r-------- 1 bob bob    27 Nov 2 13:34 file.txt
prompt$ logout

Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: alice
Password:
Last login: Thu Nov 1 17:12:23 on tty1
prompt$ cd /tmp
```

Fun setuid example

```
prompt$ echo "This is an unreadable file" > file.txt
prompt$ chmod 400 file.txt
prompt$ ls -l | grep bob
-rws--x--x 1 bob bob 47292 Nov 2 13:33 bobcat
-r-------- 1 bob bob    27 Nov 2 13:34 file.txt
prompt$ logout

Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: alice
Password:
Last login: Thu Nov 1 17:12:23 on tty1
prompt$ cd /tmp
prompt$ █
```

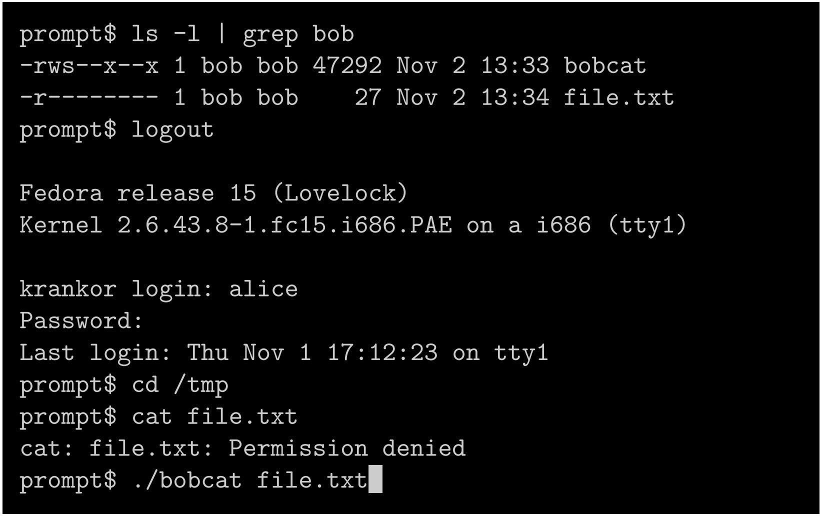
Fun setuid example

```
prompt$ echo "This is an unreadable file" > file.txt
prompt$ chmod 400 file.txt
prompt$ ls -l | grep bob
-rws--x--x 1 bob bob 47292 Nov 2 13:33 bobcat
-r-------- 1 bob bob    27 Nov 2 13:34 file.txt
prompt$ logout

Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: alice
Password:
Last login: Thu Nov 1 17:12:23 on tty1
prompt$ cd /tmp
prompt$ cat file.txt
```

Fun setuid example

```
prompt$ ls -l | grep bob
-rws--x--x 1 bob bob 47292 Nov 2 13:33 bobcat
-r-------- 1 bob bob    27 Nov 2 13:34 file.txt
prompt$ logout

Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: alice
Password:
Last login: Thu Nov 1 17:12:23 on tty1
prompt$ cd /tmp
prompt$ cat file.txt
cat: file.txt: Permission denied
prompt$ █
```

Fun setuid example



```
prompt$ ls -l | grep bob
-rws--x--x 1 bob bob 47292 Nov 2 13:33 bobcat
-r-------- 1 bob bob    27 Nov 2 13:34 file.txt
prompt$ logout

Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: alice
Password:
Last login: Thu Nov 1 17:12:23 on tty1
prompt$ cd /tmp
prompt$ cat file.txt
cat: file.txt: Permission denied
prompt$ ./bobcat file.txt
```

Fun setuid example

```
-r-------- 1 bob bob    27 Nov 2 13:34 file.txt
prompt$ logout

Fedora release 15 (Lovelock)
Kernel 2.6.43.8-1.fc15.i686.PAE on a i686 (tty1)

krankor login: alice
Password:
Last login: Thu Nov 1 17:12:23 on tty1
prompt$ cd /tmp
prompt$ cat file.txt
cat: file.txt: Permission denied
prompt$ ./bobcat file.txt
This is an unreadable file
prompt$
```

Running stuff as another user

Choices so far:

- `su [username]`
  - Actually starts a shell as the given user
  - Need to know the user's password, of course
- setuid programs
  - Executables owned by another user, with `setuid` bit set
  - Anyone with permission can run these at any time
  - Are a security concern (we will discuss why later)

Is there a way to let certain users run certain things as other users?

sudo utility

- Usage: `sudo command args args ...`
- Some important switches (check your `man` pages):

  `-u` : specify “new” user (default is `root`)
  `-g` : specify “new” group
  (default is primary of new user)
- Runs the command but sets
  - Effective userid to the specified user
  - Effective groupid to the specified group
- When run as `root`: no password prompt
- When run as ordinary user:
  - Type your own password (usually)

sudo utility

- Usage: `sudo command args args ...`
- Some important switches (check your `man` pages):

  `-u` : specify "new" user (default is `root`)
  `-g` : specify "new" group
  (default is primary of new user)

- Runs the command but sets
  - Effective userid to the specified user
  - Effective groupid to the specified group
- When run as `root`: no password prompt
- When run as ordinary user:
  - Type your own password (usually)

- Is this a huge security hole?

sudo utility

- Usage: `sudo command args args ...`
- Some important switches (check your `man` pages):

  `-u` : specify "new" user (default is `root`)
  `-g` : specify "new" group
  (default is primary of new user)
- Runs the command but sets
  - Effective userid to the specified user
  - Effective groupid to the specified group
- When run as `root`: no password prompt
- When run as ordinary user:
  - Type your own password (usually)
- Is this a huge security hole? Usually not. . .

sudo configuration

/etc/sudoers

- Specifies who may do what
- Fancy configuration file
  - Can define aliases (variables)
- Should be edited with `visudo`
  - Locks the file and edits it with `vi`
- For more info: `man sudoers`

Example fragment of `/etc/sudoers`:

```
## This is a comment
Cmnd_Alias POWER = /sbin/shutdown, /sbin/poweroff, /sbin/halt
## Allow anyone in group "staff" to shut down the machine
%staff  ALL = POWER
```

Important stuff with sudo

- Can configure sudo to log usage
  - Successful attempts
  - Unsuccessful attempts
- Can configure sudo to send mail on failed attempts

Important stuff with sudo

- Can configure sudo to log usage
  - Successful attempts
  - Unsuccessful attempts
- Can configure sudo to send mail on failed attempts

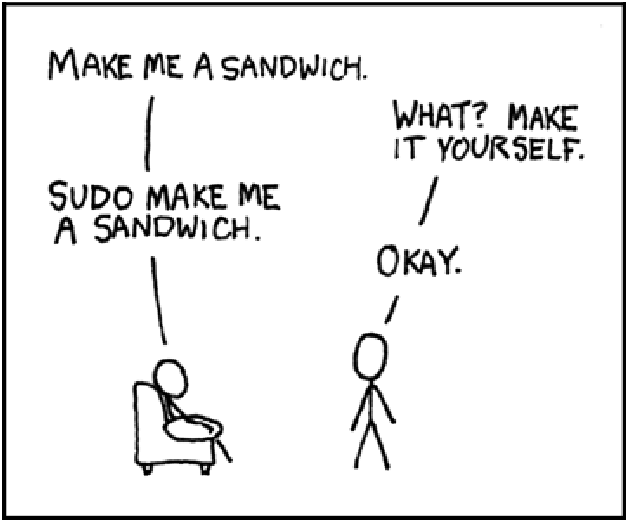
- But still need to be careful with sudo configuration:
  - `sudo bash` gives you a root shell
    - Administrator accounts can do this in Mac OS
    - sudo log entry will just show that `bash` was run
    - Commands run inside `bash` will not be logged
    - Generally, this is to be avoided

Important stuff with sudo

- Can configure sudo to log usage
  - Successful attempts
  - Unsuccessful attempts
- Can configure sudo to send mail on failed attempts
- But still need to be careful with sudo configuration:
  - `sudo bash` gives you a root shell
    - Administrator accounts can do this in Mac OS
    - `sudo` log entry will just show that `bash` was run
    - Commands run inside `bash` will not be logged
    - Generally, this is to be avoided
  - Allowing `sudo vi` is just as bad
    - `vi` lets you invoke a shell using "`:!`"
    - Not to mention the ability to edit any file

`chsh`	change your login shell
`gpasswd`	group administration
`groupadd`	add a new group
`groupdel`	remove an existing group
`groupmod`	modify an existing group
`id`	show userID and groupID
`newgrp`	change the current group
`passwd`	change passwords
`sudo`	run a command as another user
`useradd`	add a new user account
`userdel`	remove an existing user account
`usermod`	modify an existing user account

An appropriate `xkcd` comic: `http://xkcd.com/149`



End of lecture