Shell Tricks

ComS 252 — Iowa State University

Andrew Miner

Useful tidbits (1)

echo: display arguments

- May need to quote things to get spacing, symbols right
- Safe ("non-destructive") way to try shell tricks

```
prompt$
```

Useful tidbits (1)

echo: display arguments

- May need to quote things to get spacing, symbols right
- Safe ("non-destructive") way to try shell tricks

```
prompt$ echo Hello, world
```

Useful tidbits (1)

echo: display arguments

- May need to quote things to get spacing, symbols right
- Safe ("non-destructive") way to try shell tricks

```
prompt$ echo Hello, world
Hello, world
prompt$
```

Useful tidbits (1)

echo: display arguments

- May need to quote things to get spacing, symbols right
- Safe ("non-destructive") way to try shell tricks

```
prompt$ echo Hello, world
Hello, world
prompt$ echo Hello                    world
```

Useful tidbits (1)

echo: display arguments

- May need to quote things to get spacing, symbols right
- Safe ("non-destructive") way to try shell tricks

```
prompt$ echo Hello, world
Hello, world
prompt$ echo Hello                    world
Hello world
prompt$
```

Useful tidbits (1)

echo: display arguments

- May need to quote things to get spacing, symbols right
- Safe ("non-destructive") way to try shell tricks

```
prompt$ echo Hello, world
Hello, world
prompt$ echo Hello                    world
Hello world
prompt$ echo 'Hello          world'
```

Useful tidbits (1)

echo: display arguments

- May need to quote things to get spacing, symbols right
- Safe ("non-destructive") way to try shell tricks

```
prompt$ echo Hello, world
Hello, world
prompt$ echo Hello                    world
Hello world
prompt$ echo 'Hello          world'
Hello          world
prompt$
```

Useful tidbits (2)

semicolon: separate commands on a single line

```
prompt$
```

Useful tidbits (2)

semicolon: separate commands on a single line

```
prompt$ echo Hello; pwd; ls; echo world
```

Useful tidbits (2)

semicolon: separate commands on a single line

```
prompt$ echo Hello; pwd; ls; echo world
Hello
/home/alice
a.out   bar.txt   foo.txt   hello.c
world
prompt$
```

Useful tidbits (3)

`su`: substitute user

- More precisely: run a shell as another user
- Usage: `su [userid]`
- If no userid is specified, default is `root`
- You will be prompted for the user's password

Useful tidbits (3)

`su`: substitute user

- More precisely: run a shell as another user
- Usage: `su [userid]`
- If no userid is specified, default is `root`
- You will be prompted for the user's password
  - Unless you are `root`

Useful tidbits (3)

`su`: substitute user

- More precisely: run a shell as another user
- Usage: `su [userid]`
- If no userid is specified, default is `root`
- You will be prompted for the user's password
  - Unless you are `root`

`whoami`: who am I

- Give the userid for the current shell
- The prompt may not change when you do `su`

Examples for su, whoami

```
prompt$
```

Examples for su, whoami

```
prompt$ whoami
```

Examples for su, whoami

```
prompt$ whoami
alice
prompt$
```

Examples for su, whoami

```
prompt$ whoami
alice
prompt$ su
```

Examples for su, whoami

```
prompt$ whoami
alice
prompt$ su
Password:
```

I correctly type the root password here

Examples for su, whoami

```
prompt$ whoami
alice
prompt$ su
Password:
prompt$
```

Examples for su, whoami

```
prompt$ whoami
alice
prompt$ su
Password:
prompt$ su bob
```

Examples for su, whoami

```
prompt$ whoami
alice
prompt$ su
Password:
prompt$ su bob
prompt$
```

Examples for su, whoami

```
prompt$ whoami
alice
prompt$ su
Password:
prompt$ su bob
prompt$ whoami
```

Examples for su, whoami

```
prompt$ whoami
alice
prompt$ su
Password:
prompt$ su bob
prompt$ whoami
bob
prompt$
```

Examples for su, whoami

```
prompt$ whoami
alice
prompt$ su
Password:
prompt$ su bob
prompt$ whoami
bob
prompt$ exit
```

Examples for su, whoami

```
prompt$ whoami
alice
prompt$ su
Password:
prompt$ su bob
prompt$ whoami
bob
prompt$ exit
prompt$
```

Examples for su, whoami

```
prompt$ whoami
alice
prompt$ su
Password:
prompt$ su bob
prompt$ whoami
bob
prompt$ exit
prompt$ whoami
```

Examples for su, whoami

```
prompt$ whoami
alice
prompt$ su
Password:
prompt$ su bob
prompt$ whoami
bob
prompt$ exit
prompt$ whoami
root
prompt$
```

History

- The shell keeps a history of the last N commands executed
  - N can be configured, usually around 1,000
- You can edit and re-run previous commands

`history`: display command history

```
prompt$
```

History

- The shell keeps a history of the last N commands executed
  - N can be configured, usually around 1,000
- You can edit and re-run previous commands

`history`: display command history

```
prompt$ history
```

History

- The shell keeps a history of the last *N* commands executed
  - *N* can be configured, usually around 1,000
- You can edit and re-run previous commands

`history`: display command history

```
500 ls -l foo.txt
501 chmod 777 /tmp
502 chmod +t /tmp
503 ls -alF /tmp
prompt$
```

Searching history

```
prompt$
```

- Up arrow: move earlier in history
- Down arrow: move later in history
- Ctrl–r: reverse search
- Left and right arrows: edit the command
- Enter: execute the line

Searching history

```
prompt$ history
```

- Up arrow: move earlier in history
- Down arrow: move later in history
- Ctrl–r: reverse search
- Left and right arrows: edit the command
- Enter: execute the line

Searching history

```
498 ls -l
499 cat foo.txt
500 ls -l foo.txt
501 chmod 777 /tmp
502 chmod +t /tmp
503 ls -alF /tmp
prompt$
```

- Up arrow: move earlier in history
- Down arrow: move later in history
- Ctrl–r: reverse search
- Left and right arrows: edit the command
- Enter: execute the line

Searching history

```
498 ls -l
499 cat foo.txt
500 ls -l foo.txt
501 chmod 777 /tmp
502 chmod +t /tmp
503 ls -alF /tmp
prompt$
```

- Up arrow: move earlier in history
- Down arrow: move later in history
- Ctrl–r: reverse search
- Left and right arrows: edit the command
- Enter: execute the line

Searching history

```
498 ls -l
499 cat foo.txt
500 ls -l foo.txt
501 chmod 777 /tmp
502 chmod +t /tmp
503 ls -alF /tmp
prompt$ ls -alF /tmp
```

- Up arrow: move earlier in history
- Down arrow: move later in history
- Ctrl–r: reverse search
- Left and right arrows: edit the command
- Enter: execute the line

Searching history

```
498 ls -l
499 cat foo.txt
500 ls -l foo.txt
501 chmod 777 /tmp
502 chmod +t /tmp
503 ls -alF /tmp
prompt$ chmod +t /tmp
```

- Up arrow: move earlier in history
- Down arrow: move later in history
- Ctrl–r: reverse search
- Left and right arrows: edit the command
- Enter: execute the line

Searching history

```
498 ls -l
499 cat foo.txt
500 ls -l foo.txt
501 chmod 777 /tmp
502 chmod +t /tmp
503 ls -alF /tmp
prompt$ chmod 777 /tmp
```

- Up arrow: move earlier in history
- Down arrow: move later in history
- Ctrl–r: reverse search
- Left and right arrows: edit the command
- Enter: execute the line

Searching history

```
498 ls -l
499 cat foo.txt
500 ls -l foo.txt
501 chmod 777 /tmp
502 chmod +t /tmp
503 ls -alF /tmp
prompt$ ls -l foo.txt
```

- Up arrow: move earlier in history
- Down arrow: move later in history
- Ctrl–r: reverse search
- Left and right arrows: edit the command
- Enter: execute the line

Searching history

```
498 ls -l
499 cat foo.txt
500 ls -l foo.txt
501 chmod 777 /tmp
502 chmod +t /tmp
503 ls -alF /tmp
prompt$ ls -l foo.txt
```

- Up arrow: move earlier in history
- Down arrow: move later in history
- Ctrl–r: reverse search
- Left and right arrows: edit the command
- Enter: execute the line

Searching history

```
498 ls -l
499 cat foo.txt
500 ls -l foo.txt
501 chmod 777 /tmp
502 chmod +t /tmp
503 ls -alF /tmp
prompt$ ls -l foo.txt
```

- Up arrow: move earlier in history
- Down arrow: move later in history
- Ctrl–r: reverse search
- Left and right arrows: edit the command
- Enter: execute the line

Searching history

```
498 ls -l
499 cat foo.txt
500 ls -l foo.txt
501 chmod 777 /tmp
502 chmod +t /tmp
503 ls -alF /tmp
prompt$ chmod 777 /tmp
```

- Up arrow: move earlier in history
- Down arrow: move later in history
- Ctrl–r: reverse search
- Left and right arrows: edit the command
- Enter: execute the line

Searching history

```
498 ls -l
499 cat foo.txt
500 ls -l foo.txt
501 chmod 777 /tmp
502 chmod +t /tmp
503 ls -alF /tmp
prompt$ chmod +t /tmp
```

- Up arrow: move earlier in history
- Down arrow: move later in history
- Ctrl–r: reverse search
- Left and right arrows: edit the command
- Enter: execute the line

Searching history

```
498 ls -l
499 cat foo.txt
500 ls -l foo.txt
501 chmod 777 /tmp
502 chmod +t /tmp
503 ls -alF /tmp
prompt$ ls -alF /tmp
```

- Up arrow: move earlier in history
- Down arrow: move later in history
- Ctrl–r: reverse search
- Left and right arrows: edit the command
- Enter: execute the line

Searching history

```
498 ls -l
499 cat foo.txt
500 ls -l foo.txt
501 chmod 777 /tmp
502 chmod +t /tmp
503 ls -alF /tmp
prompt$ ls -alF /tmp
```

- Up arrow: move earlier in history
- Down arrow: move later in history
- Ctrl–r: reverse search
- Left and right arrows: edit the command
- Enter: execute the line

Searching history

```
498 ls -l
499 cat foo.txt
500 ls -l foo.txt
501 chmod 777 /tmp
502 chmod +t /tmp
503 ls -alF /tmp
(reverse-i-search)‘’: ls -alF /tmp
```

- Up arrow: move earlier in history
- Down arrow: move later in history
- Ctrl–r: reverse search
- Left and right arrows: edit the command
- Enter: execute the line

Searching history

```
498 ls -l
499 cat foo.txt
500 ls -l foo.txt
501 chmod 777 /tmp
502 chmod +t /tmp
503 ls -alF /tmp
(reverse-i-search)`cat': at foo.txt
```

- Up arrow: move earlier in history
- Down arrow: move later in history
- Ctrl–r: reverse search
- Left and right arrows: edit the command
- Enter: execute the line

Searching history

```
498 ls -l
499 cat foo.txt
500 ls -l foo.txt
501 chmod 777 /tmp
502 chmod +t /tmp
503 ls -alF /tmp
prompt$ cat foo.txt
```

- Up arrow: move earlier in history
- Down arrow: move later in history
- Ctrl–r: reverse search
- Left and right arrows: edit the command
- Enter: execute the line

Searching history

```
498 ls -l
499 cat foo.txt
500 ls -l foo.txt
501 chmod 777 /tmp
502 chmod +t /tmp
503 ls -alF /tmp
prompt$ cat foo.txt
```

- Up arrow: move earlier in history
- Down arrow: move later in history
- Ctrl–r: reverse search
- Left and right arrows: edit the command
- Enter: execute the line

Searching history

```
498 ls -l
499 cat foo.txt
500 ls -l foo.txt
501 chmod 777 /tmp
502 chmod +t /tmp
503 ls -alF /tmp
prompt$ cat bar.txt foo.txt
```

- Up arrow: move earlier in history
- Down arrow: move later in history
- Ctrl–r: reverse search
- Left and right arrows: edit the command
- Enter: execute the line

Searching history

```
498 ls -l
499 cat foo.txt
500 ls -l foo.txt
501 chmod 777 /tmp
502 chmod +t /tmp
503 ls -alF /tmp
prompt$ cat bar.txt foo.txt
```

- Up arrow: move earlier in history
- Down arrow: move later in history
- Ctrl–r: reverse search
- Left and right arrows: edit the command
- Enter: execute the line

Searching history

```
501 chmod 777 /tmp
502 chmod +t /tmp
503 ls -alF /tmp
prompt$ cat bar.txt foo.txt
cat: bar.txt: No such file or directory
cat: foo.txt: No such file or directory
prompt$
```

- Up arrow: move earlier in history
- Down arrow: move later in history
- Ctrl–r: reverse search
- Left and right arrows: edit the command
- Enter: execute the line

Bang (!) substitution

`!!` : the previous command

`!pre` : the most recent command starting with "pre"

`!`n : the nth command in the history

`!$` : last argument from previous command

`!*` : all arguments from previous command

`!!:`n : n^{th} argument from previous command

You can use both command and argument selection:

`!cat:$` : last arg. from last "cat" command

`!402:3` : third argument from command 402 in history

Examples: bang substitution

```
prompt$
```

Examples: bang substitution

```
prompt$ history
```

Examples: bang substitution

```
495 cat bar.txt
496 rm loop
497 mv bar.txt foo.txt
498 ls -l
499 cat foo.txt
500 ls -l foo.txt
501 chmod 777 /tmp
502 chmod +t /tmp
503 ls -alF /tmp
504 history
505 cat bar.txt foo.txt
prompt$
```

Examples: bang substitution

```
495 cat bar.txt
496 rm loop
497 mv bar.txt foo.txt
498 ls -l
499 cat foo.txt
500 ls -l foo.txt
501 chmod 777 /tmp
502 chmod +t /tmp
503 ls -alF /tmp
504 history
505 cat bar.txt foo.txt
prompt$ echo !!
```

Examples: bang substitution

```
498 ls -l
499 cat foo.txt
500 ls -l foo.txt
501 chmod 777 /tmp
502 chmod +t /tmp
503 ls -alF /tmp
504 history
505 cat bar.txt foo.txt
prompt$ echo !!
echo history
history
prompt$
```

Examples: bang substitution

```
498 ls -l
499 cat foo.txt
500 ls -l foo.txt
501 chmod 777 /tmp
502 chmod +t /tmp
503 ls -alF /tmp
504 history
505 cat bar.txt foo.txt
prompt$ echo !!
echo history
history
prompt$ echo !c; echo !ch; echo !ch:1; echo !503:0
```

Examples: bang substitution

```
504 history
505 cat bar.txt foo.txt
prompt$ echo !!
echo history
history
prompt$ echo !c; echo !ch; echo !ch:1; echo !503:0
echo cat bar.txt foo.txt; echo chmod +t /tmp; echo +t;...
cat bar.txt foo.txt
chmod +t /tmp
+t
ls
prompt$
```

Examples: bang substitution

```
504 history
505 cat bar.txt foo.txt
prompt$ echo !!
echo history
history
prompt$ echo !c; echo !ch; echo !ch:1; echo !503:0
echo cat bar.txt foo.txt; echo chmod +t /tmp; echo +t;...
cat bar.txt foo.txt
chmod +t /tmp
+t
ls
prompt$ !503:0
```

Examples: bang substitution

```
echo history
history
prompt$ echo !c; echo !ch; echo !ch:1; echo !503:0
echo cat bar.txt foo.txt; echo chmod +t /tmp; echo +t;...
cat bar.txt foo.txt
chmod +t /tmp
+t
ls
prompt$ !503:0
ls
a.out   bar.txt   foo.txt   hello.c
prompt$
```

Tab completion

If you press Tab in the shell

- The shell will try to complete a pathname after the cursor
- If there are multiple pathnames, it will fill in as much as it can
- Pressing Tab again will show the list of possible completions

```
prompt$
```

(Normal typing)

Tab completion

If you press Tab in the shell

- The shell will try to complete a pathname after the cursor
- If there are multiple pathnames, it will fill in as much as it can
- Pressing Tab again will show the list of possible completions

```
prompt$ ls
```

(Normal typing)

Tab completion

If you press Tab in the shell

- The shell will try to complete a pathname after the cursor
- If there are multiple pathnames, it will fill in as much as it can
- Pressing Tab again will show the list of possible completions

```
prompt$ ls
hello.cc        hello.h         some.crazy.file
prompt$
```

(Normal typing)

Tab completion

If you press Tab in the shell

- The shell will try to complete a pathname after the cursor
- If there are multiple pathnames, it will fill in as much as it can
- Pressing Tab again will show the list of possible completions

```
prompt$ ls
hello.cc        hello.h         some.crazy.file
prompt$ cat so
```

(Press Tab)

Tab completion

If you press Tab in the shell

- The shell will try to complete a pathname after the cursor
- If there are multiple pathnames, it will fill in as much as it can
- Pressing Tab again will show the list of possible completions

```
prompt$ ls
hello.cc        hello.h        some.crazy.file
prompt$ cat some.crazy.file
```

(The shell completed the path name; back to normal typing)

Tab completion

If you press Tab in the shell

- The shell will try to complete a pathname after the cursor
- If there are multiple pathnames, it will fill in as much as it can
- Pressing Tab again will show the list of possible completions

```
prompt$ ls
hello.cc        hello.h         some.crazy.file
prompt$ cat some.crazy.file
This is a crazy file.  What did you expect?
prompt$
```

(Normal typing)

Tab completion

If you press Tab in the shell

- The shell will try to complete a pathname after the cursor
- If there are multiple pathnames, it will fill in as much as it can
- Pressing Tab again will show the list of possible completions

```
prompt$ ls
hello.cc        hello.h         some.crazy.file
prompt$ cat some.crazy.file
This is a crazy file.  What did you expect?
prompt$ cp /etc/fs
```

(Press Tab)

Tab completion

If you press Tab in the shell

- The shell will try to complete a pathname after the cursor
- If there are multiple pathnames, it will fill in as much as it can
- Pressing Tab again will show the list of possible completions

```
prompt$ ls
hello.cc        hello.h         some.crazy.file
prompt$ cat some.crazy.file
This is a crazy file.  What did you expect?
prompt$ cp /etc/fstab
```

(The shell completed the path name; back to normal typing)

Tab completion

If you press Tab in the shell

- The shell will try to complete a pathname after the cursor
- If there are multiple pathnames, it will fill in as much as it can
- Pressing Tab again will show the list of possible completions

```
prompt$ ls
hello.cc        hello.h         some.crazy.file
prompt$ cat some.crazy.file
This is a crazy file.  What did you expect?
prompt$ cp /etc/fstab fstab.mine
```

Tab completion

If you press Tab in the shell

- The shell will try to complete a pathname after the cursor
- If there are multiple pathnames, it will fill in as much as it can
- Pressing Tab again will show the list of possible completions

```
prompt$ ls
hello.cc        hello.h         some.crazy.file
prompt$ cat some.crazy.file
This is a crazy file.  What did you expect?
prompt$ cp /etc/fstab fstab.mine
prompt$
```

(Normal typing)

Tab completion

If you press Tab in the shell

- The shell will try to complete a pathname after the cursor
- If there are multiple pathnames, it will fill in as much as it can
- Pressing Tab again will show the list of possible completions

```
prompt$ ls
hello.cc        hello.h         some.crazy.file
prompt$ cat some.crazy.file
This is a crazy file.  What did you expect?
prompt$ cp /etc/fstab fstab.mine
prompt$ rm h
```

(Press Tab)

Tab completion

If you press Tab in the shell

- The shell will try to complete a pathname after the cursor
- If there are multiple pathnames, it will fill in as much as it can
- Pressing Tab again will show the list of possible completions

```
prompt$ ls
hello.cc        hello.h         some.crazy.file
prompt$ cat some.crazy.file
This is a crazy file.  What did you expect?
prompt$ cp /etc/fstab fstab.mine
prompt$ rm hello.
```

(That's all the shell can complete; press Tab again)

Tab completion

If you press Tab in the shell

- The shell will try to complete a pathname after the cursor
- If there are multiple pathnames, it will fill in as much as it can
- Pressing Tab again will show the list of possible completions

```
prompt$ ls
hello.cc        hello.h         some.crazy.file
prompt$ cat some.crazy.file
This is a crazy file.  What did you expect?
prompt$ cp /etc/fstab fstab.mine
prompt$ rm hello.
hello.cc  hello.h
prompt$ rm hello.
```

(Normal typing)

Tab completion

If you press Tab in the shell

- The shell will try to complete a pathname after the cursor
- If there are multiple pathnames, it will fill in as much as it can
- Pressing Tab again will show the list of possible completions

```
prompt$ ls
hello.cc        hello.h         some.crazy.file
prompt$ cat some.crazy.file
This is a crazy file.  What did you expect?
prompt$ cp /etc/fstab fstab.mine
prompt$ rm hello.
hello.cc  hello.h
prompt$ rm hello.h
```

Tab completion

If you press Tab in the shell

- The shell will try to complete a pathname after the cursor
- If there are multiple pathnames, it will fill in as much as it can
- Pressing Tab again will show the list of possible completions

```
prompt$ ls
hello.cc        hello.h         some.crazy.file
prompt$ cat some.crazy.file
This is a crazy file.  What did you expect?
prompt$ cp /etc/fstab fstab.mine
prompt$ rm hello.
hello.cc  hello.h
prompt$ rm hello.h
prompt$
```

Globbing

- The shell allows arguments with "wildcards"
- Specifying wildcards:

  `?` : fill in any one character
  `*` : fill in any characters (zero or more)
  `[list]` : fill in any character from the list

- The shell will replace the argument with matching path names
- If there are no matching path names:
  - Depends on the shell
  - Might give an error
  - Might leave the argument with the wildcard characters
- Fun fact
  - This is implemented in a C library function, `glob()`
  - See `man 3 glob` for more info
  - There are similar modules for other languages, e.g., Python

Example: globbing (bash in Linux)

```
prompt$
```

Example: globbing (bash in Linux)

```
prompt$ ls
```

Example: globbing (bash in Linux)

```
prompt$ ls
bat.c    cat.c    catfood  eat.o    what.o
bat.h    cat.o    eat.c    makefile zip.c
bat.o    catch    eat.h    what.c   zip.o
prompt$
```

Example: globbing (bash in Linux)

```
prompt$ ls
bat.c    cat.c    catfood  eat.o    what.o
bat.h    cat.o    eat.c    makefile zip.c
bat.o    catch    eat.h    what.c   zip.o
prompt$ echo ?at.h
```

Example: globbing (bash in Linux)

```
prompt$ ls
bat.c    cat.c    catfood  eat.o    what.o
bat.h    cat.o    eat.c    makefile zip.c
bat.o    catch    eat.h    what.c   zip.o
prompt$ echo ?at.h
bat.h eat.h
prompt$
```

Example: globbing (bash in Linux)

```
prompt$ ls
bat.c    cat.c    catfood  eat.o    what.o
bat.h    cat.o    eat.c    makefile zip.c
bat.o    catch    eat.h    what.c   zip.o
prompt$ echo ?at.h
bat.h eat.h
prompt$ ls *at.?
```

Example: globbing (bash in Linux)

```
prompt$ ls
bat.c    cat.c    catfood  eat.o    what.o
bat.h    cat.o    eat.c    makefile zip.c
bat.o    catch    eat.h    what.c   zip.o
prompt$ echo ?at.h
bat.h eat.h
prompt$ ls *at.?
bat.c  bat.o  cat.o  eat.h  what.c
bat.h  cat.c  eat.c  eat.o  what.o
prompt$
```

Example: globbing (bash in Linux)

```
prompt$ ls
bat.c    cat.c    catfood  eat.o    what.o
bat.h    cat.o    eat.c    makefile zip.c
bat.o    catch    eat.h    what.c   zip.o
prompt$ echo ?at.h
bat.h eat.h
prompt$ ls *at.?
bat.c  bat.o  cat.o  eat.h  what.c
bat.h  cat.c  eat.c  eat.o  what.o
prompt$ rm *.o
```

Example: globbing (bash in Linux)

```
prompt$ ls
bat.c    cat.c    catfood  eat.o    what.o
bat.h    cat.o    eat.c    makefile zip.c
bat.o    catch    eat.h    what.c   zip.o
prompt$ echo ?at.h
bat.h eat.h
prompt$ ls *at.?
bat.c  bat.o  cat.o  eat.h  what.c
bat.h  cat.c  eat.c  eat.o  what.o
prompt$ rm *.o
prompt$
```

Example: globbing (bash in Linux)

```
prompt$ ls
bat.c    cat.c    catfood  eat.o    what.o
bat.h    cat.o    eat.c    makefile zip.c
bat.o    catch    eat.h    what.c   zip.o
prompt$ echo ?at.h
bat.h eat.h
prompt$ ls *at.?
bat.c  bat.o  cat.o  eat.h  what.c
bat.h  cat.c  eat.c  eat.o  what.o
prompt$ rm *.o
prompt$ echo cat??
```

Example: globbing (bash in Linux)

```
prompt$ ls
bat.c    cat.c    catfood  eat.o    what.o
bat.h    cat.o    eat.c    makefile zip.c
bat.o    catch    eat.h    what.c   zip.o
prompt$ echo ?at.h
bat.h eat.h
prompt$ ls *at.?
bat.c  bat.o  cat.o  eat.h  what.c
bat.h  cat.c  eat.c  eat.o  what.o
prompt$ rm *.o
prompt$ echo cat??
cat.c catch
prompt$
```

Example: globbing (bash in Linux)

```
prompt$ ls
bat.c    cat.c    catfood  eat.o    what.o
bat.h    cat.o    eat.c    makefile zip.c
bat.o    catch    eat.h    what.c   zip.o
prompt$ echo ?at.h
bat.h eat.h
prompt$ ls *at.?
bat.c  bat.o  cat.o  eat.h  what.c
bat.h  cat.c  eat.c  eat.o  what.o
prompt$ rm *.o
prompt$ echo cat??
cat.c catch
prompt$ echo [be]at*
```

Example: globbing (bash in Linux)

```
prompt$ ls
bat.c    cat.c    catfood  eat.o    what.o
bat.h    cat.o    eat.c    makefile zip.c
bat.o    catch    eat.h    what.c   zip.o
prompt$ echo ?at.h
bat.h eat.h
prompt$ ls *at.?
bat.c  bat.o  cat.o  eat.h  what.c
bat.h  cat.c  eat.c  eat.o  what.o
prompt$ rm *.o
prompt$ echo cat??
cat.c catch
prompt$ echo [be]at*
bat.c bat.h eat.c eat.h
prompt$
```

Example: globbing (bash in Linux)

```
prompt$ ls
bat.c    cat.c    catfood  eat.o    what.o
bat.h    cat.o    eat.c    makefile zip.c
bat.o    catch    eat.h    what.c   zip.o
prompt$ echo ?at.h
bat.h eat.h
prompt$ ls *at.?
bat.c  bat.o  cat.o  eat.h  what.c
bat.h  cat.c  eat.c  eat.o  what.o
prompt$ rm *.o
prompt$ echo cat??
cat.c catch
prompt$ echo [be]at*
bat.c bat.h eat.c eat.h
prompt$ rm ?
```

Example: globbing (bash in Linux)

```
prompt$ ls
bat.c    cat.c    catfood  eat.o    what.o
bat.h    cat.o    eat.c    makefile zip.c
bat.o    catch    eat.h    what.c   zip.o
prompt$ echo ?at.h
bat.h eat.h
prompt$ ls *at.?
bat.c  bat.o  cat.o  eat.h  what.c
bat.h  cat.c  eat.c  eat.o  what.o
prompt$ rm *.o
prompt$ echo cat??
cat.c catch
prompt$ echo [be]at*
bat.c bat.h eat.c eat.h
prompt$ rm ?
rm: No match.
prompt$
```

Tricky example

```
prompt$
```

Tricky example

```
prompt$ ls
```

Tricky example

```
prompt$ ls
-l       wtf.txt
prompt$
```

Tricky example

```
prompt$ ls
-l       wtf.txt
prompt$ ls *
```

Tricky example

```
prompt$ ls
-l       wtf.txt
prompt$ ls *
-rw------- 1 alice hackers   12 Jun 11 2003 wtf.txt
prompt$
```

Tricky example

```
prompt$ ls
-l       wtf.txt
prompt$ ls *
-rw------- 1 alice hackers   12 Jun 11 2003 wtf.txt
prompt$ echo ls *
```

Tricky example

```
prompt$ ls
-l       wtf.txt
prompt$ ls *
-rw------- 1 alice hackers   12 Jun 11 2003 wtf.txt
prompt$ echo ls *
ls -l wtf.txt
prompt$
```

Tricky example

```
prompt$ ls
-l       wtf.txt
prompt$ ls *
-rw------- 1 alice hackers   12 Jun 11 2003 wtf.txt
prompt$ echo ls *
ls -l wtf.txt
prompt$ rm -l
```

Tricky example

```
prompt$ ls
-l       wtf.txt
prompt$ ls *
-rw------- 1 alice hackers   12 Jun 11 2003 wtf.txt
prompt$ echo ls *
ls -l wtf.txt
prompt$ rm -l
rm: invalid option -- 'l'
Try `rm --help' for more information.
prompt$
```

Tricky example

```
prompt$ ls
-l       wtf.txt
prompt$ ls *
-rw------- 1 alice hackers   12 Jun 11 2003 wtf.txt
prompt$ echo ls *
ls -l wtf.txt
prompt$ rm -l
rm: invalid option -- 'l'
Try 'rm --help' for more information.
prompt$ rm -- -l
```

Tricky example

```
prompt$ ls
-l       wtf.txt
prompt$ ls *
-rw------- 1 alice hackers   12 Jun 11 2003 wtf.txt
prompt$ echo ls *
ls -l wtf.txt
prompt$ rm -l
rm: invalid option -- 'l'
Try `rm --help' for more information.
prompt$ rm -- -l
prompt$
```

Evil file names

- In UNIX, file names can contain any character except /
- Even spaces.
  - Need to quote the name, e.g., `"file name"`
  - Or "escape" the space, e.g., `file\ name`
- Some file names are just plain evil
- Example: a file named "`-fr ~`"
  - Why is this evil?

Evil file names

- In UNIX, file names can contain any character except /
- Even spaces.
  - Need to quote the name, e.g., `"file name"`
  - Or "escape" the space, e.g., `file\ name`
- Some file names are just plain evil
- Example: a file named "`-fr ~`"
  - Why is this evil?
  - What does `rm -fr ~` do?

Evil file names

- In UNIX, file names can contain any character except /
- Even spaces.
  - Need to quote the name, e.g., `"file name"`
  - Or "escape" the space, e.g., `file\ name`
- Some file names are just plain evil
- Example: a file named "`-fr ~`"
  - Why is this evil?
  - What does `rm -fr ~` do?
  - Ok, but I know to use `rm -- "-fr ~"` so I'm fine.
  - Right?

Evil file names

- In UNIX, file names can contain any character except /
- Even spaces.
  - Need to quote the name, e.g., `"file name"`
  - Or "escape" the space, e.g., `file\ name`
- Some file names are just plain evil
- Example: a file named "`-fr ~`"
  - Why is this evil?
  - What does `rm -fr ~` do?
  - Ok, but I know to use `rm -- "-fr ~"` so I'm fine.
  - Right?
  - What if you want to remove the directory, and do `rm *`

Smart file name guidelines

- Avoid file names with shell symbols
  - E.g.: ?, *, ~
- Avoid file names containing spaces
- Avoid file names starting with -
- Yes, it is possible to deal with all of these
- But why make your life more difficult on purpose?

Foreground execution

- When you execute a command, you get another prompt

Foreground execution

- When you execute a command, you get another prompt
  - after the command finishes executing

Foreground execution

- When you execute a command, you get another prompt
  - after the command finishes executing
- This is what "foreground execution" means

Foreground execution

- When you execute a command, you get another prompt
  - after the command finishes executing
- This is what "foreground execution" means
- For example, if I run `ls`:

```
prompt$
```

Foreground execution

- When you execute a command, you get another prompt
  - after the command finishes executing
- This is what "foreground execution" means
- For example, if I run `ls`:

```
prompt$ ls
```

Foreground execution

- When you execute a command, you get another prompt
  - after the command finishes executing
- This is what "foreground execution" means
- For example, if I run `ls`:

```
prompt$ ls
bat.c    cat.c    catfood  eat.o    what.o
bat.h    cat.o    eat.c    makefile zip.c
bat.o    catch    eat.h    what.c   zip.o
prompt$
```

`ls` finishes, then we get the prompt

Foreground execution

- When you execute a command, you get another prompt
  - after the command finishes executing
- This is what "foreground execution" means
- For example, if I run `ls`:

```
prompt$ ls
bat.c    cat.c    catfood  eat.o    what.o
bat.h    cat.o    eat.c    makefile zip.c
bat.o    catch    eat.h    what.c   zip.o
prompt$
```

  `ls` finishes, then we get the prompt
- This is more dramatic when we use the GUI...

Terminal

```
prompt$
```

Terminal

```
prompt$ firefox
```

Ter

```
prompt$ firefox
```

xkcd: Formal Languages

www.xkcd.org/1090/

ARCHIVE
WHAT IF?
BLAG
STORE
ABOUT

xkcd

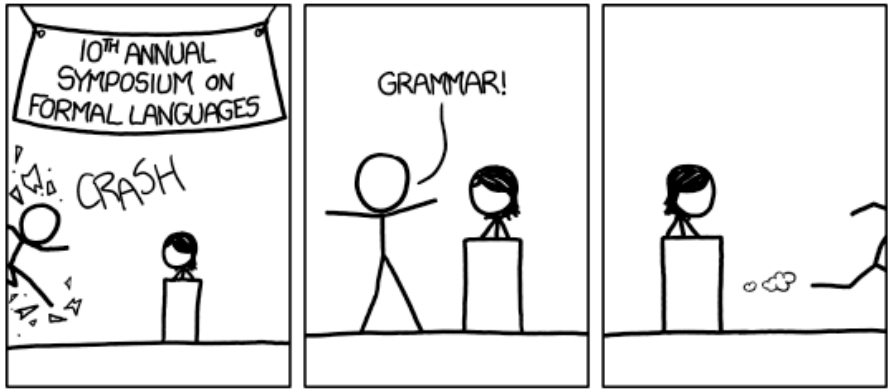
A WEBCOMIC OF ROMANCE, SARCASM, MATH, AND LANGUAGE.

WHAT-IF.XKCD.COM

THIS WEEK'S QUESTION: WHAT IF THE GLASS WERE *REALLY* HALF EMPTY?

FORMAL LANGUAGES

I< < PREV RANDOM NEXT > >I



I< < PREV RANDOM NEXT > >I

PERMANENT LINK TO THIS COMIC: HTTP://XKCD.COM/1090/
IMAGE URL (FOR HOTLINKING/EMBEDDING): HTTP://IMGS.XKCD.COM/COMICS/FORMAL_LANGUAGES.PNG

Ter

```
prompt$ firefox
```

xkcd: Server Problem

www.xkcd.org/1084/

ARCHIVE
WHAT IF?
BLAG
STORE
ABOUT

xkcd

A WEBCOMIC OF ROMANCE, SARCASM, MATH, AND LANGUAGE.

WHAT-IF.XKCD.COM

THIS WEEK'S QUESTION: WHAT IF THE GLASS WERE *REALLY* HALF EMPTY?

SERVER PROBLEM

I< < PREV RANDOM NEXT > >I

I< < PREV RANDOM NEXT > >I

Terminal

```
prompt$ firefox
prompt$
```

Job control

- The shell offers much more control over execution than we have seen so far
- A command sent to be executed is called a job
- In a single shell, we can
  - Suspend, resume, and terminate jobs
  - Have several jobs running, simultaneously
  - Check the status of our jobs

Terminating, suspending, and resuming jobs

Foreground mode

Ctrl–c : Terminate the foreground job

- Usually works...
- The job is interrupted, and destroyed
- Memory is freed
- We get the prompt back

Ctrl–z : Suspend the foreground job

- Usually works...
- The job is interrupted, but not destroyed
- In a GUI, window may not redraw itself
- The job is still in memory
- We can resume the job later if we want
- We get the prompt back

%*n* : Resume job *n* (in foreground mode)

- The job number is given when you suspend it

Terminal

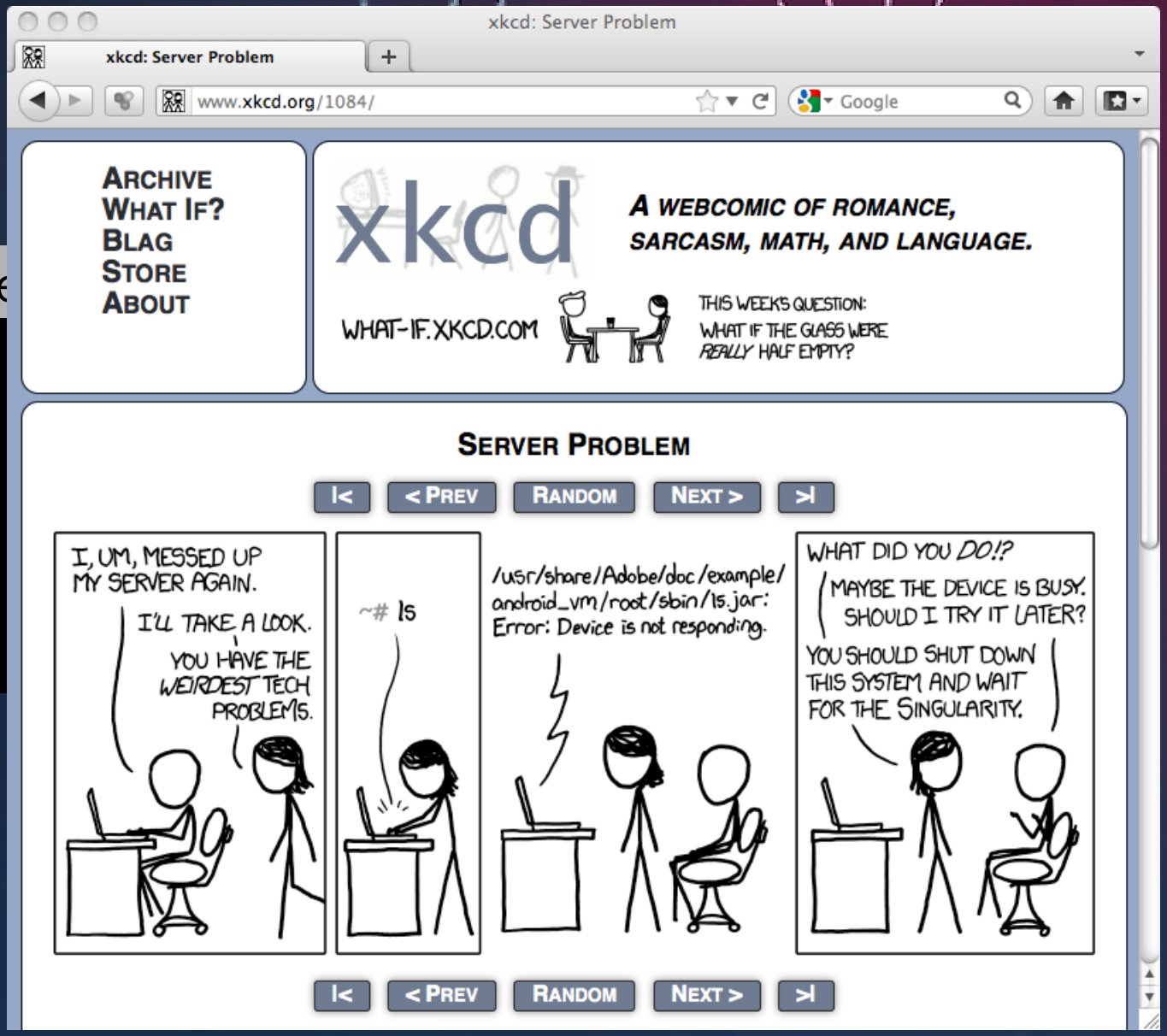
```
prompt$
```

Terminal

```
prompt$ firefox
```

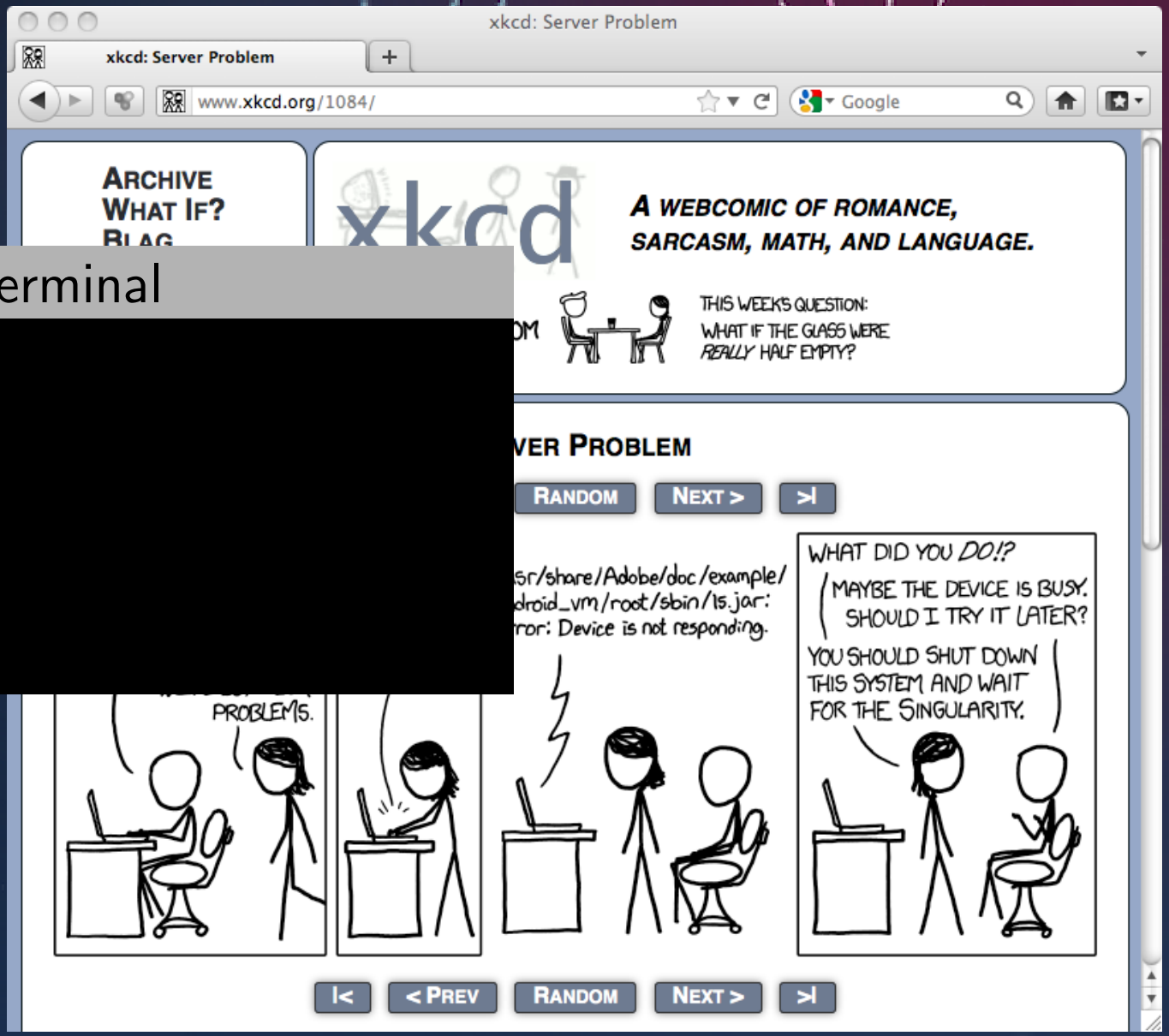
Te

```
prompt$ firefox
```



Terminal

```
prompt$ firefox
```



Terminal

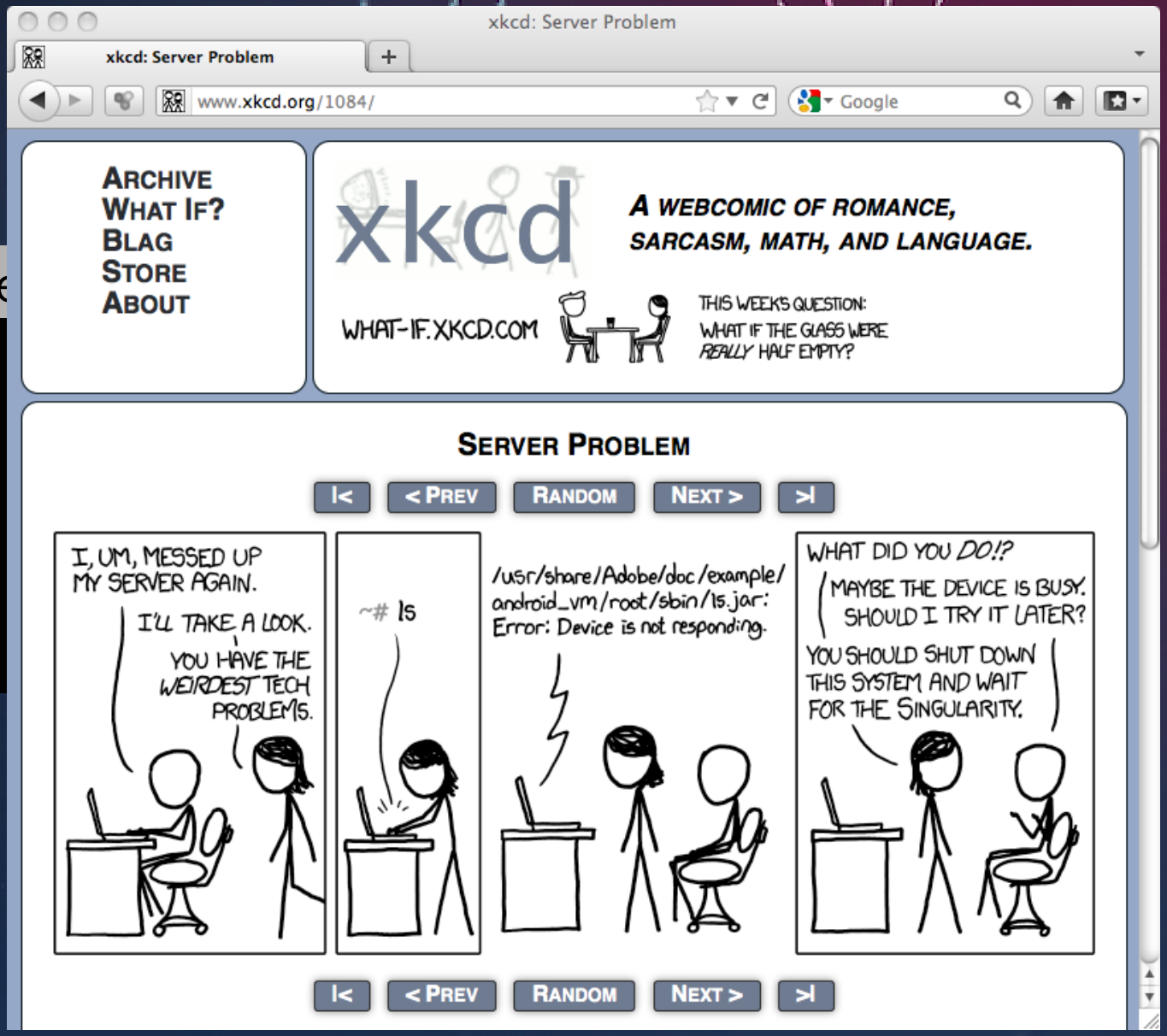
```
prompt$ firefox
^C
prompt$
```

Terminal

```
prompt$ firefox
^C
prompt$ firefox
```

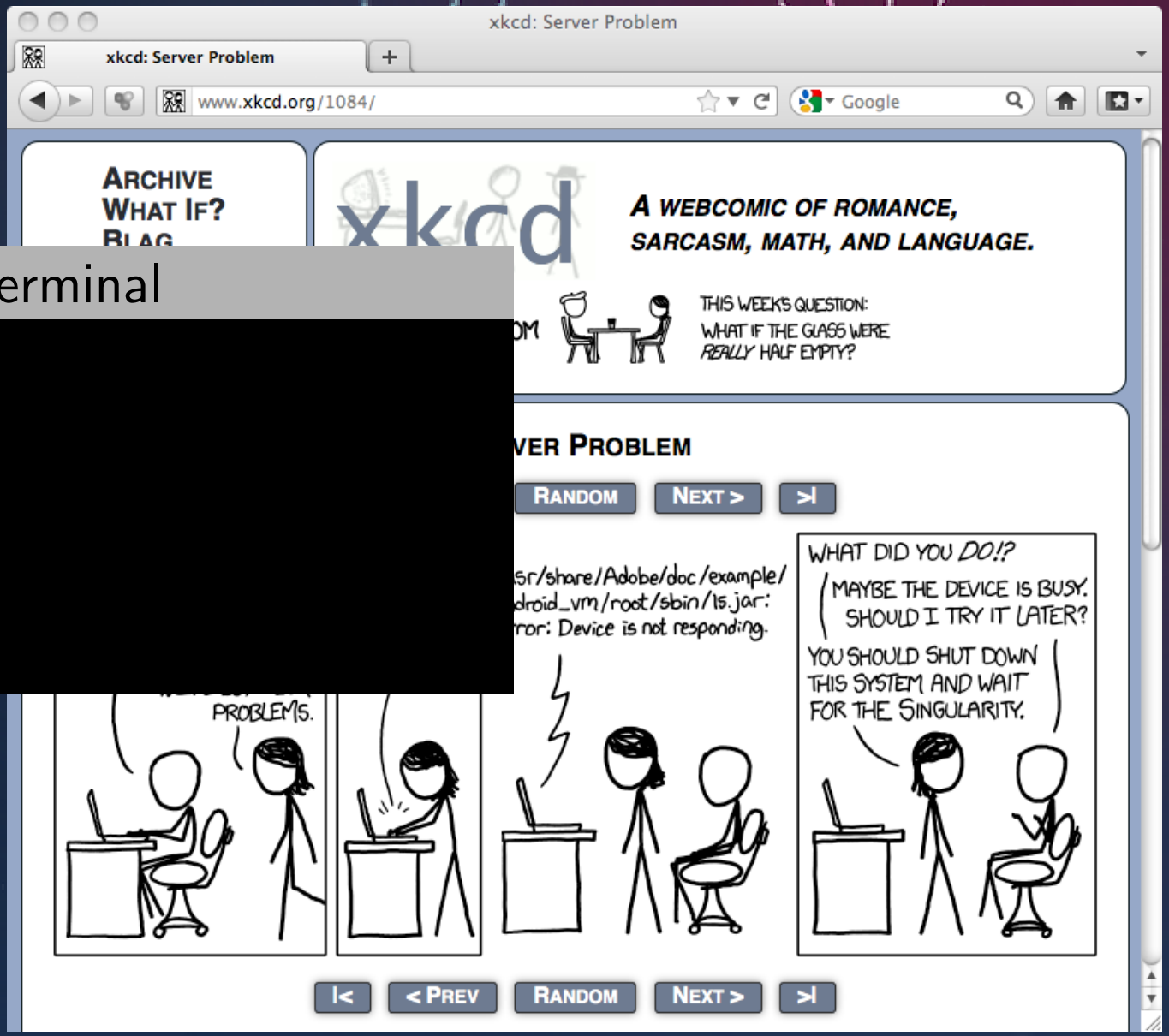
Te

```
prompt$ firefox
^C
prompt$ firefox
```



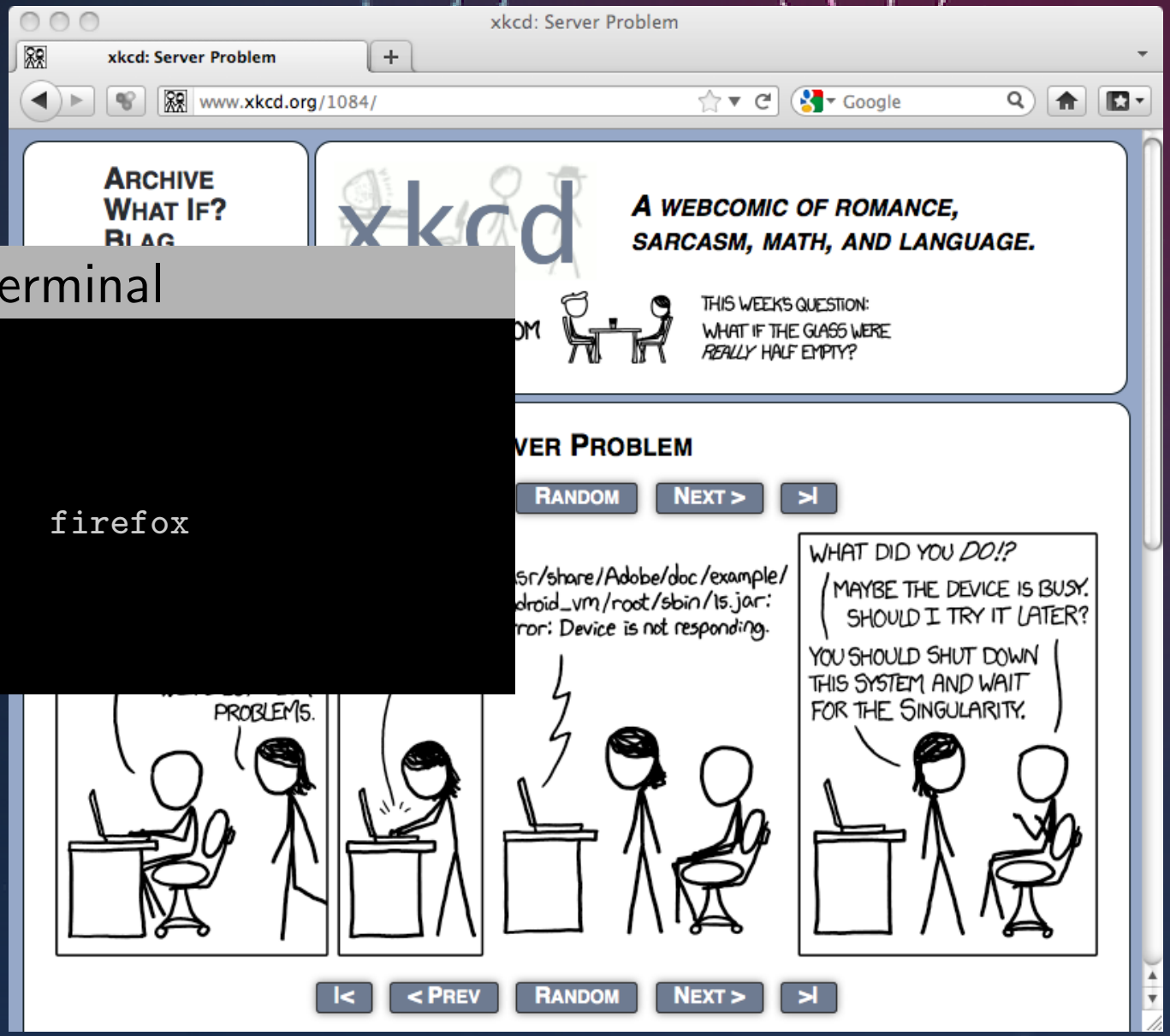
Terminal

```
prompt$ firefox
^C
prompt$ firefox
```



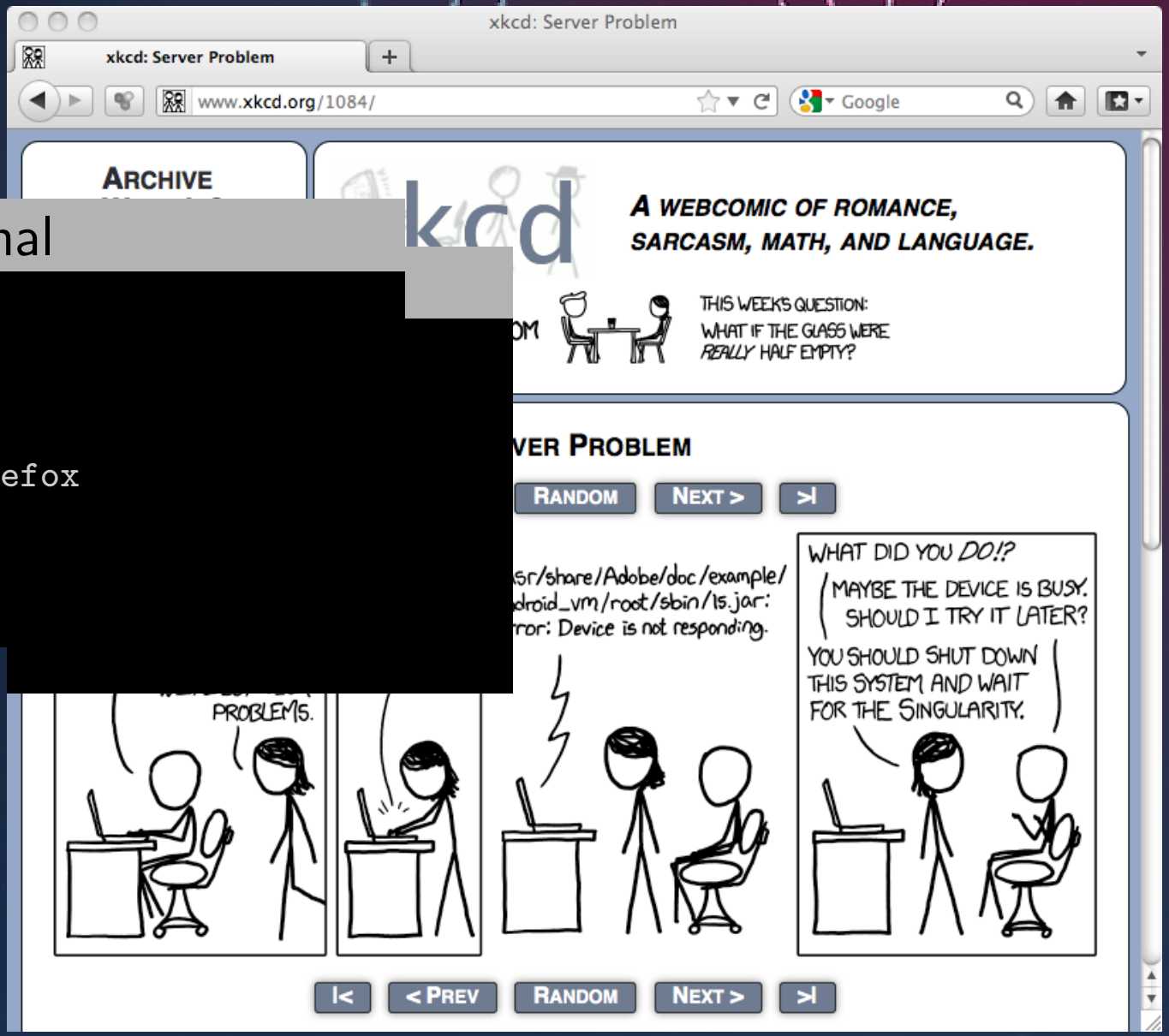
Terminal

```
prompt$ firefox
^C
prompt$ firefox
^Z
[1]+  Stopped                 firefox
prompt$
```



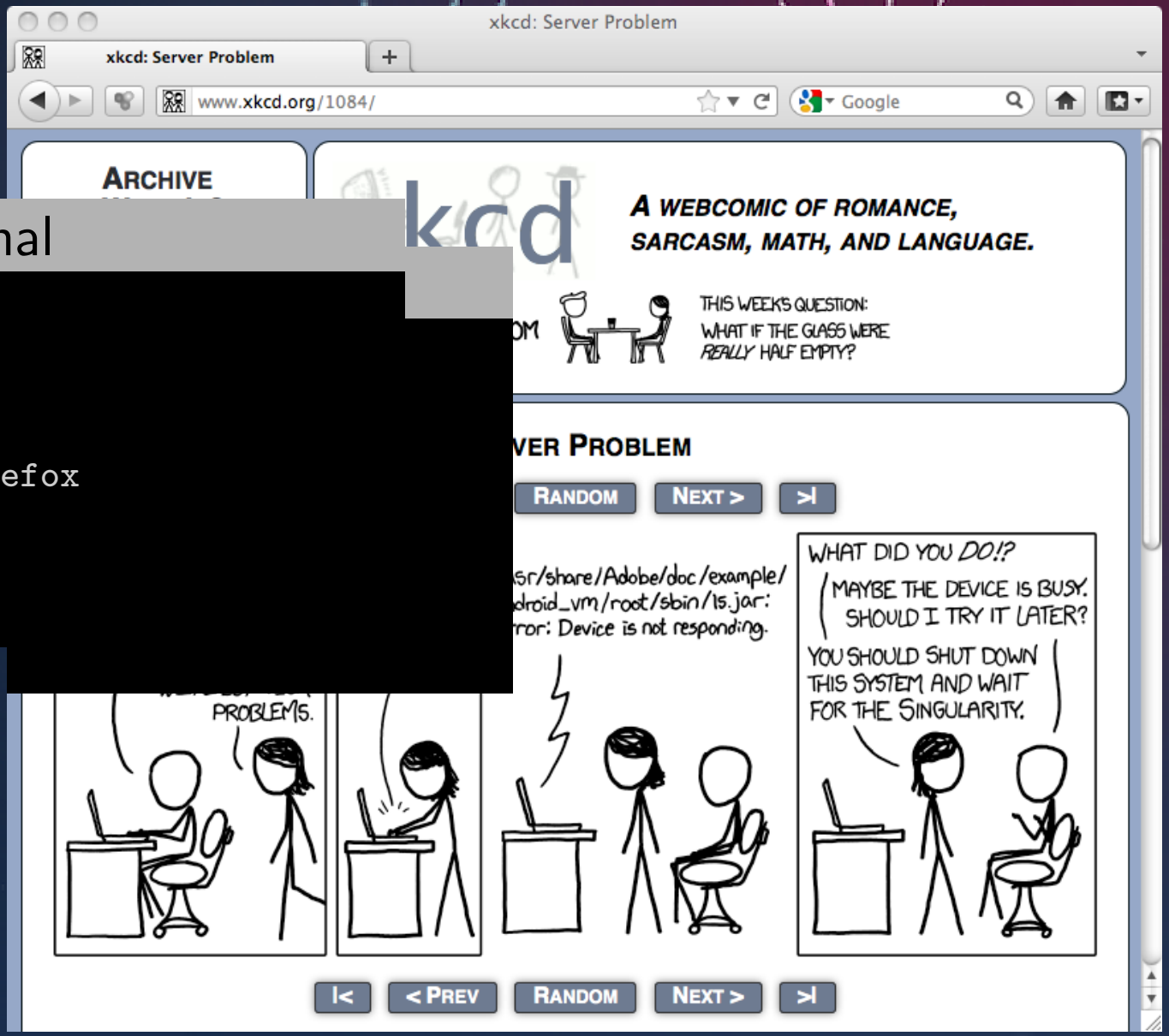
Terminal

```
prompt$ firefox
^C
prompt$  firefox
^Z
[1]+  Stopped                 firefox
prompt$
```



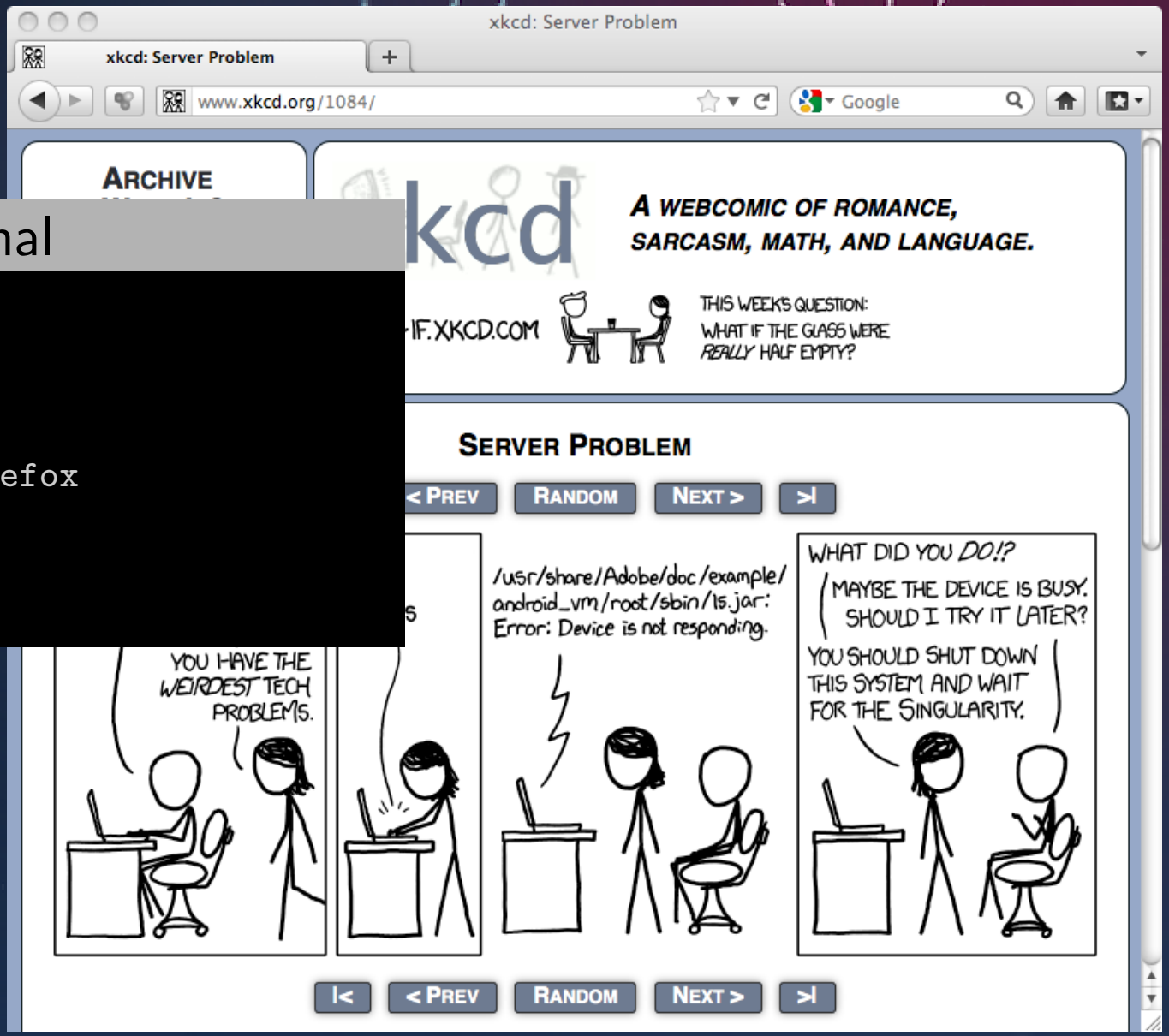
Terminal

```
prompt$ firefox
^C
prompt$  firefox
^Z
[1]+  Stopped                 firefox
prompt$ %1
```



Terminal

```
prompt$ firefox
^C
prompt$  firefox
^Z
[1]+  Stopped                 firefox
prompt$ %1
firefox
```



Background execution

- Backround mode: get the prompt back immediately
  - Shell does not wait for the command to finish
  - The command keeps executing after the prompt is back

Background execution

- Backround mode: get the prompt back immediately
  - Shell does not wait for the command to finish
  - The command keeps executing after the prompt is back
- To do this: type your command followed by "&"
  - Shell displays the job number for the command

Background execution

- Backround mode: get the prompt back immediately
  - Shell does not wait for the command to finish
  - The command keeps executing after the prompt is back
- To do this: type your command followed by "&"
  - Shell displays the job number for the command
- For example:

```
prompt$
```

Background execution

- Backround mode: get the prompt back immediately
  - Shell does not wait for the command to finish
  - The command keeps executing after the prompt is back
- To do this: type your command followed by "&"
  - Shell displays the job number for the command
- For example:

```
prompt$ cp fc17.iso /mnt/usbstick &
```

Background execution

- Backround mode: get the prompt back immediately
  - Shell does not wait for the command to finish
  - The command keeps executing after the prompt is back
- To do this: type your command followed by "&"
  - Shell displays the job number for the command
- For example:

```
prompt$ cp fc17.iso /mnt/usbstick &
[1]+ Running        cp
prompt$
```

Background execution

- Backround mode: get the prompt back immediately
  - Shell does not wait for the command to finish
  - The command keeps executing after the prompt is back
- To do this: type your command followed by "&"
  - Shell displays the job number for the command
- For example:

```
prompt$ cp fc17.iso /mnt/usbstick &
[1]+ Running        cp
prompt$ ls -l /mnt/usbstick
```

Background execution

- Backround mode: get the prompt back immediately
  - Shell does not wait for the command to finish
  - The command keeps executing after the prompt is back
- To do this: type your command followed by "&"
  - Shell displays the job number for the command
- For example:

```
prompt$ cp fc17.iso /mnt/usbstick &
[1]+ Running        cp
prompt$ ls -l /mnt/usbstick
-rw-------  1  alice  staff   134217728  Sep 13 12:56  fc17.iso
prompt$
```

Background execution

- Backround mode: get the prompt back immediately
  - Shell does not wait for the command to finish
  - The command keeps executing after the prompt is back
- To do this: type your command followed by "&"
  - Shell displays the job number for the command
- For example:

```
prompt$ cp fc17.iso /mnt/usbstick &
[1]+ Running        cp
prompt$ ls -l /mnt/usbstick
-rw-------  1  alice  staff   134217728  Sep 13 12:56  fc17.iso
prompt$ !!
```

Background execution

- Backround mode: get the prompt back immediately
  - Shell does not wait for the command to finish
  - The command keeps executing after the prompt is back
- To do this: type your command followed by "&"
  - Shell displays the job number for the command
- For example:

```
prompt$ cp fc17.iso /mnt/usbstick &
[1]+ Running        cp
prompt$ ls -l /mnt/usbstick
-rw-------  1  alice  staff   134217728  Sep 13 12:56  fc17.iso
prompt$ !!
ls -l /mnt/usbstick
-rw-------  1  alice  staff   268435456  Sep 13 12:57  fc17.iso
prompt$
```

Job control

`jobs`: display jobs for this shell

- Lists jobs by job number
- Gives status for each job

`%n` or `fg n` or `fg %n` : Run job *n* in the foreground

- Works for jobs that are suspended, or running in the background

`%n&` or `bg n` or `bg %n` : Run job *n* in the background

- Works for jobs that are suspended, or running in the background
  - But not needed if the job is already running in the background

Example: job control

```
prompt$
```

Example: job control

```
prompt$ !ls
```

Example: job control

```
prompt$ !ls
ls -l /mnt/usbstick
-rw-------  1  alice  staff  1073741824  Sep 13 13:04  fc17.iso
prompt$
```

Example: job control

```
prompt$ !ls
ls -l /mnt/usbstick
-rw-------  1  alice  staff  1073741824  Sep 13 13:04  fc17.iso
prompt$ jobs
```

Example: job control

```
prompt$ !ls
ls -l /mnt/usbstick
-rw-------  1  alice  staff  1073741824  Sep 13 13:04  fc17.iso
prompt$ jobs
[1]+  Running   cp fc17.iso /mnt/usbstick
prompt$
```

Example: job control

```
prompt$ !ls
ls -l /mnt/usbstick
-rw-------  1  alice  staff  1073741824  Sep 13 13:04  fc17.iso
prompt$ jobs
[1]+  Running   cp fc17.iso /mnt/usbstick
prompt$ fg 1
```

Example: job control

```
prompt$ !ls
ls -l /mnt/usbstick
-rw-------  1  alice  staff  1073741824  Sep 13 13:04  fc17.iso
prompt$ jobs
[1]+  Running   cp fc17.iso /mnt/usbstick
prompt$ fg 1
cp fc17.iso /mnt/usbstick
```

Example: job control

```
prompt$ !ls
ls -l /mnt/usbstick
-rw-------  1  alice  staff  1073741824  Sep 13 13:04  fc17.iso
prompt$ jobs
[1]+  Running   cp fc17.iso /mnt/usbstick
prompt$ fg 1
cp fc17.iso /mnt/usbstick
^Z
[1]+  Stopped   cp fc17.iso /mnt/usbstick
prompt$
```

Example: job control

```
prompt$ !ls
ls -l /mnt/usbstick
-rw-------  1  alice  staff  1073741824  Sep 13 13:04  fc17.iso
prompt$ jobs
[1]+  Running   cp fc17.iso /mnt/usbstick
prompt$ fg 1
cp fc17.iso /mnt/usbstick
^Z
[1]+  Stopped   cp fc17.iso /mnt/usbstick
prompt$ bg %1
```

Example: job control

```
prompt$ !ls
ls -l /mnt/usbstick
-rw-------  1  alice  staff  1073741824  Sep 13 13:04  fc17.iso
prompt$ jobs
[1]+  Running   cp fc17.iso /mnt/usbstick
prompt$ fg 1
cp fc17.iso /mnt/usbstick
^Z
[1]+  Stopped   cp fc17.iso /mnt/usbstick
prompt$ bg %1
[1]+ cp fc17.iso /mnt/usbstick
prompt$
```

I/O and background execution

What happens to background job output?

- Output goes as usual
- Might have several jobs writing to the terminal, simultaneously
- For each job, output will be "in order"
- Output from different jobs may be "interleaved"
  - Depends on how kernel schedules those jobs

What happens to background job input?

- A background job will suspend if it requires terminal input

Interleaving example

```
prompt$
```

Interleaving example

```
prompt$ ./myprogA
```

Interleaving example

```
prompt$ ./myprogA
a1
a2
a3
a4
prompt$
```

Interleaving example

```
prompt$ ./myprogA
a1
a2
a3
a4
prompt$ ./myprogB
```

Interleaving example

```
prompt$ ./myprogA
a1
a2
a3
a4
prompt$ ./myprogB
     b00
     b01
     b10
     b11
prompt$
```

Interleaving example

```
prompt$ ./myprogA
a1
a2
a3
a4
prompt$ ./myprogB
     b00
     b01
     b10
     b11
prompt$ ./myprogA & ./myprogB &
```

Interleaving example

```
     b10
     b11
prompt$ ./myprogA & ./myprogB &
[2] Running    ./myprogA
[3] Running    ./myprogB
a1
     b00
     b01
prompt$ a2
a3
a4
     b10
     b11
```

Interleaving example

```
     b10
     b11
prompt$ ./myprogA & ./myprogB &
[2] Running    ./myprogA
[3] Running    ./myprogB
a1
     b00
     b01
prompt$ a2
a3
a4
     b10
     b11
jobs
```

Interleaving example

```
[3] Running    ./myprogB
a1
     b00
     b01
prompt$ a2
a3
a4
     b10
     b11
jobs
[1]   Running     cp fc17.iso /mnt/usbstick
[2]-  Done        ./myprogA
[3]+  Done        ./myprogB
prompt$
```

Input example

```
prompt$
```

Input example

```
prompt$ ./hello
```

Input example

```
prompt$ ./hello
What is your name?
```

Input example

```
prompt$ ./hello
What is your name?
Bob
```

Input example

```
prompt$ ./hello
What is your name?
Bob
Hello, Bob!
prompt$
```

Input example

```
prompt$ ./hello
What is your name?
Bob
Hello, Bob!
prompt$ ./hello &
```

Input example

```
prompt$ ./hello
What is your name?
Bob
Hello, Bob!
prompt$ ./hello &
[2]+  Running     ./hello
prompt$ What is your name?
```

Input example

```
prompt$ ./hello
What is your name?
Bob
Hello, Bob!
prompt$ ./hello &
[2]+  Running     ./hello
prompt$ What is your name?
Bob
```

Input example

```
prompt$ ./hello
What is your name?
Bob
Hello, Bob!
prompt$ ./hello &
[2]+  Running     ./hello
prompt$ What is your name?
Bob
-bash:  Bob:  command not found

[2]+  Stopped     ./hello
prompt$
```

Input example

```
prompt$ ./hello
What is your name?
Bob
Hello, Bob!
prompt$ ./hello &
[2]+  Running     ./hello
prompt$ What is your name?
Bob
-bash:  Bob:  command not found

[2]+  Stopped     ./hello
prompt$ fg 2
```

Input example

```
prompt$ ./hello
What is your name?
Bob
Hello, Bob!
prompt$ ./hello &
[2]+  Running     ./hello
prompt$ What is your name?
Bob
-bash:  Bob:  command not found

[2]+  Stopped     ./hello
prompt$ fg 2
./hello
```

Input example

```
prompt$ ./hello
What is your name?
Bob
Hello, Bob!
prompt$ ./hello &
[2]+  Running     ./hello
prompt$ What is your name?
Bob
-bash:  Bob:  command not found

[2]+  Stopped     ./hello
prompt$ fg 2
./hello
Doctor Evil
```

Input example

```
prompt$ ./hello
What is your name?
Bob
Hello, Bob!
prompt$ ./hello &
[2]+  Running     ./hello
prompt$ What is your name?
Bob
-bash:  Bob:  command not found

[2]+  Stopped     ./hello
prompt$ fg 2
./hello
Doctor Evil
Hello, Doctor Evil!
prompt$
```

One last trick

`wait`: wait for jobs

Usage: `wait [%job] [%job] ...`

- Wait until *all* specified background jobs have finished
- If no jobs are specified, waits for all jobs
- `wait %n` is not quite the same as `fg %n`

One last trick

`wait`: wait for jobs

Usage: `wait [%job] [%job] ...`

- Wait until *all* specified background jobs have finished
- If no jobs are specified, waits for all jobs
- `wait %n` is not quite the same as `fg %n`
  - `wait` just waits
  - `wait` does not connect terminal input to a job
  - `wait` does not resume a stopped job

Signals?

- We can send signals to running programs
- Programs can catch and react to (some) signals
  - Programs may handle signals however they want
    - Including: ignore the signal
  - To learn how to do this in C: `man sigaction`

Signals?

- We can send signals to running programs
- Programs can catch and react to (some) signals
  - Programs may handle signals however they want
    - Including: ignore the signal
  - To learn how to do this in C: `man sigaction`

Some important signals (names often prefixed with SIG)

`TERM` : Terminate — can be caught

- Ask the program to terminate itself
- Allows programs to save information first
- No guarantee program will in fact terminate

`KILL` : Kill — cannot be caught or ignored

- The kernel terminates the program
- No warning for the program

How to send signals

`kill`: signal a job

Usage:

1. `kill -l`
   - Lists the available signals
2. `kill [-signal] %`*n*
   - Send signal to job *n*
   - Default signal is: `TERM`

- Note: `kill` can send any signal, not just `KILL`
- Yes, the choice of name is unfortunate
  - Many signals are not intended to stop a program

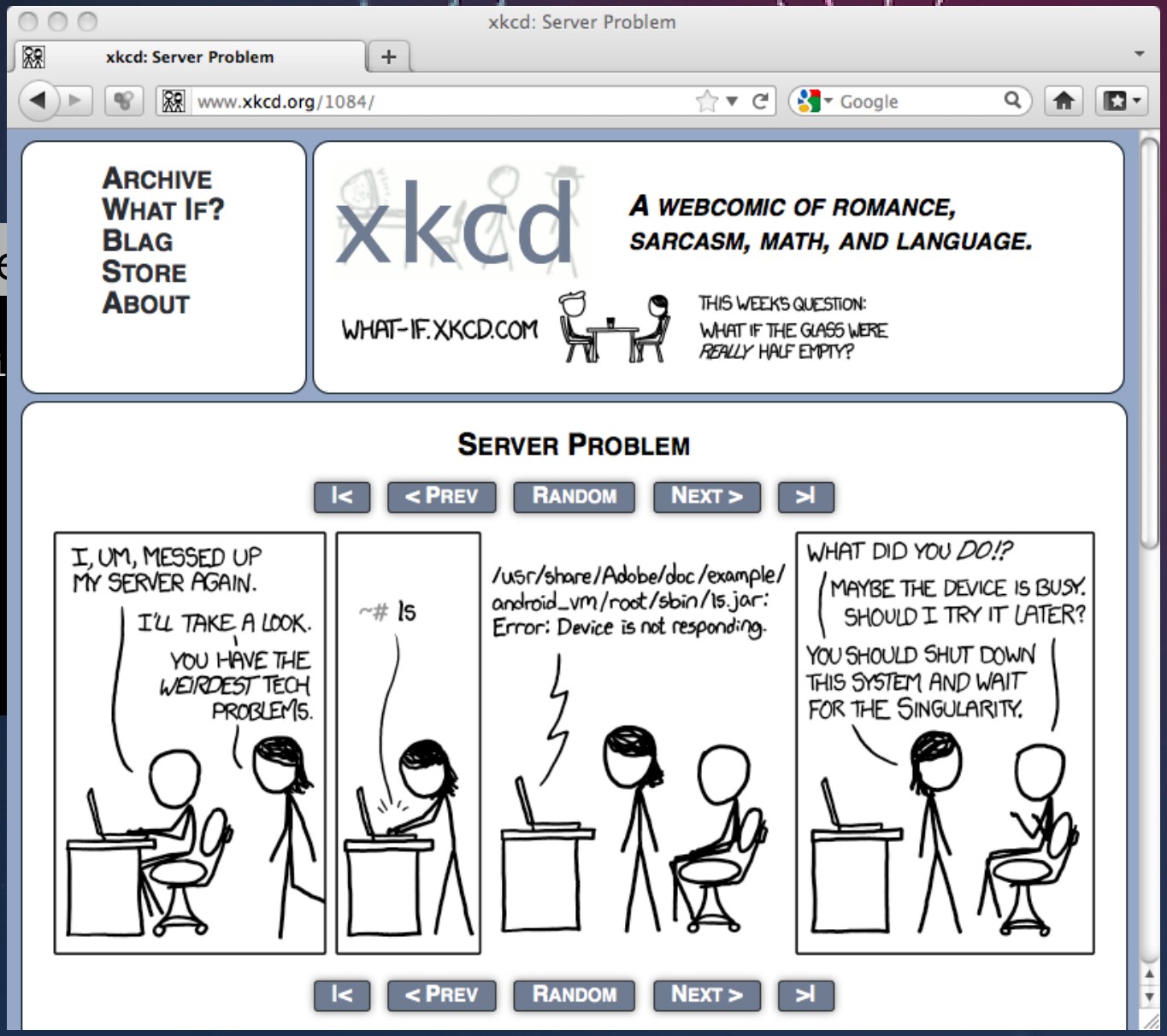
Terminal

```
prompt$
```

Terminal

```
prompt$ firefox &
```

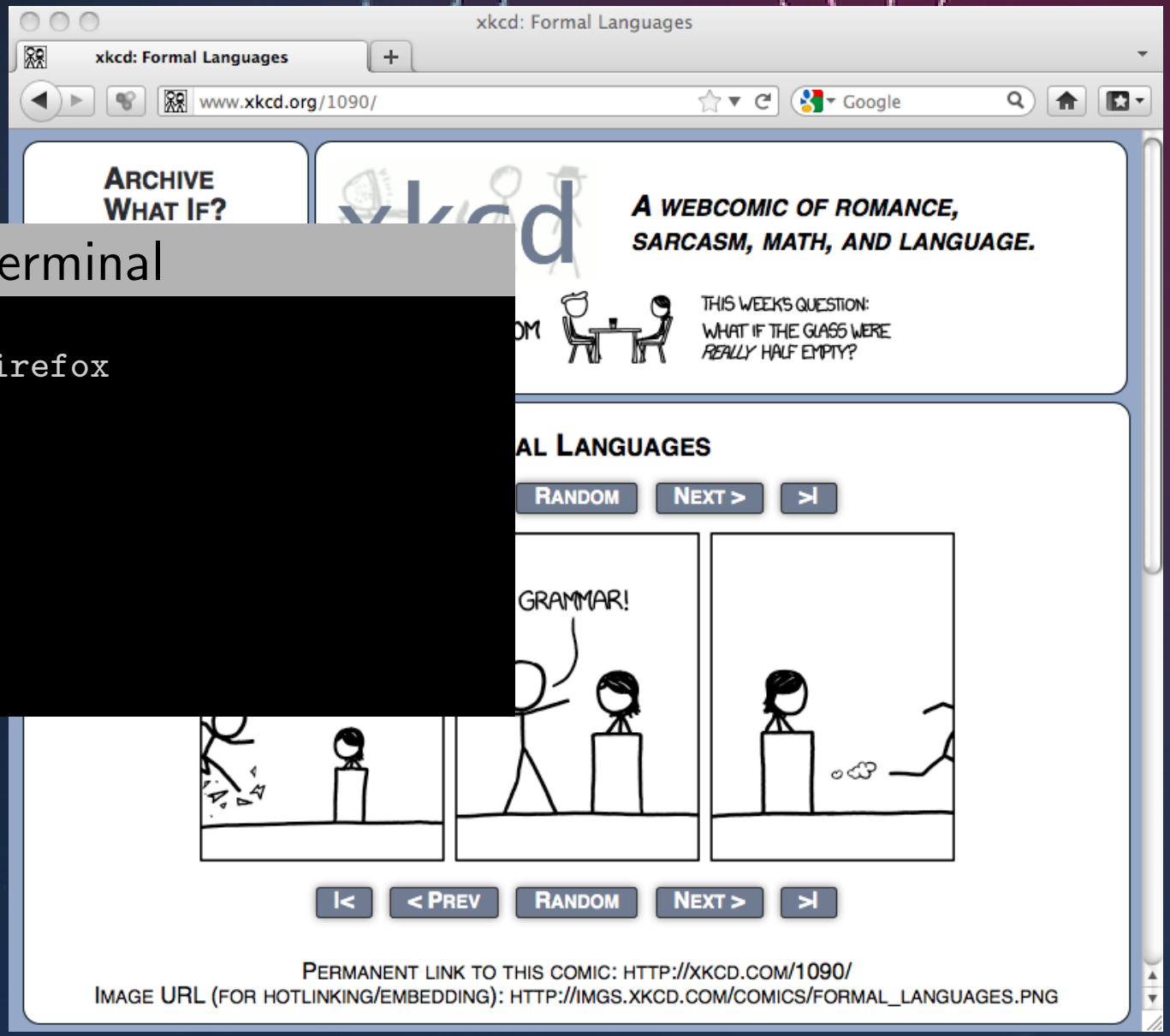
```
prompt$ firefox &
[1]+ Running      fi
prompt$
```



```
prompt$ firefox &
[1]+ Running      fi
prompt$
```

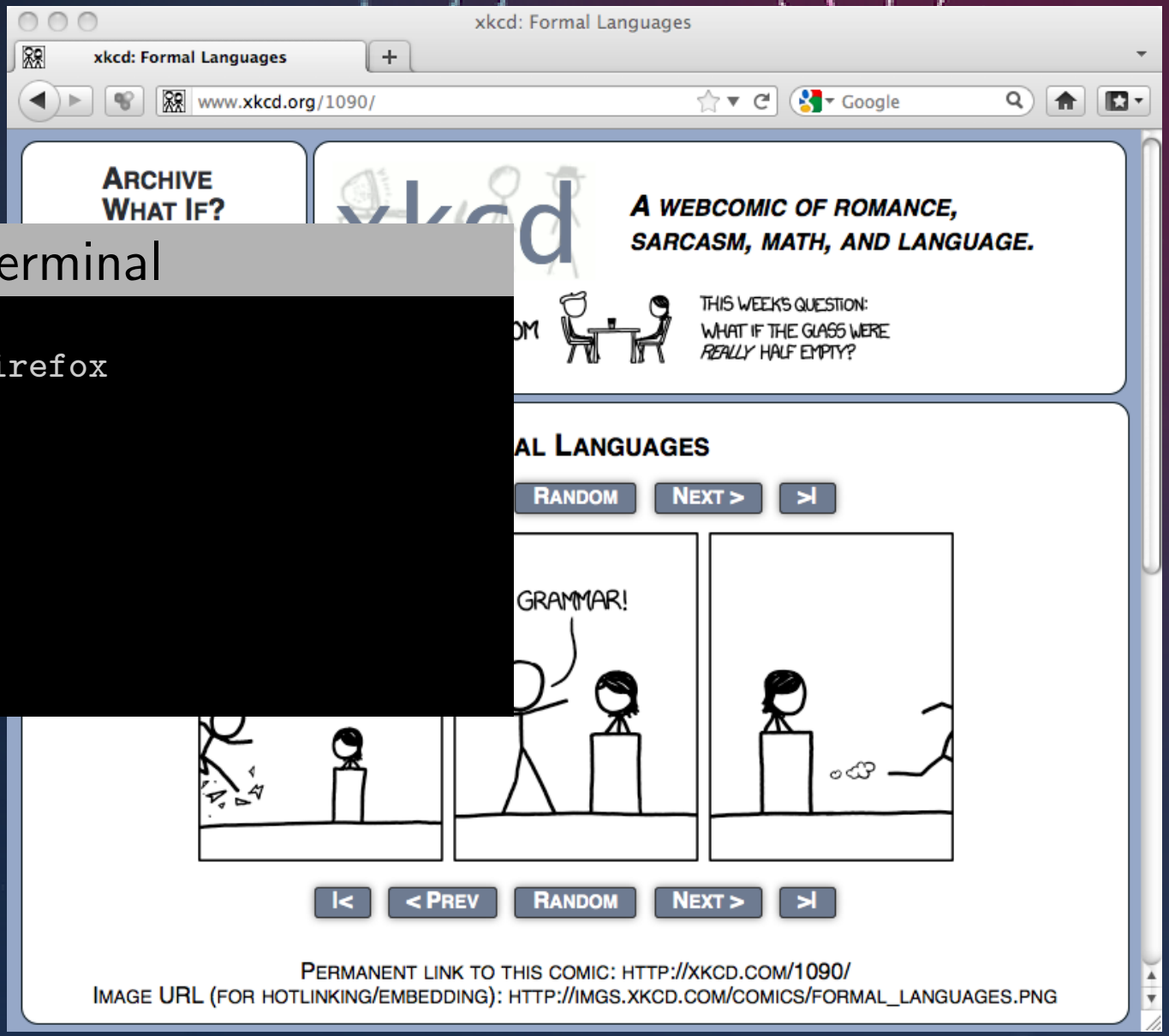
Terminal

```
prompt$ firefox &
[1]+ Running     firefox
prompt$
```



Terminal

```
prompt$ firefox &
[1]+ Running      firefox
prompt$ kill %1
```



Terminal

```
prompt$ firefox &
[1]+ Running     firefox
prompt$ kill %1
prompt$
```

Terminal

```
prompt$ firefox &
[1]+ Running     firefox
prompt$ kill %1
prompt$ firefox &
```

```
prompt$ firefox &
[1]+ Running      fi
prompt$ kill %1
prompt$ firefox &
[2]+ Running      fi
[1]  Terminated   fi
prompt$
```

```
prompt$ firefox &
[1]+ Running      fi
prompt$ kill %1
prompt$ firefox &
[2]+ Running      fi
[1]  Terminated   fi
prompt$
```

```
prompt$ firefox &
[1]+ Running      fi
prompt$ kill %1
prompt$ firefox &
[2]+ Running      fi
[1]  Terminated   fi
prompt$
```

Terminal

```
prompt$ firefox &
[1]+ Running      firefox
prompt$ kill %1
prompt$ firefox &
[2]+ Running      firefox
[1]  Terminated   firefox
prompt$
```

Why I Believe Printers Were Sent From Hell To Make Us Miserable – The Oatmeal

Why I Believe Printers Were Sen...

theoatmeal.com/comics/printers

The Oatmeal

Comics Book NEW! Quizzes Blog Shop

Most Popular All Cats Grammar Food Animals Tech

rs

t From

Hell

To Make Us Miserable

By The Oatmeal http://theoatmeal.com

Terminal

```
prompt$ firefox &
[1]+ Running      firefox
prompt$ kill %1
prompt$ firefox &
[2]+ Running      firefox
[1]  Terminated   firefox
prompt$ kill %2
```

Terminal

```
prompt$ firefox &
[1]+ Running      firefox
prompt$ kill %1
prompt$ firefox &
[2]+ Running      firefox
[1]  Terminated   firefox
prompt$ kill %2
prompt$
```

Why I Believe Printers Were Sent From Hell To Make Us Miserable – The Oatmeal

Why I Believe Printers Were Sen...

theoatmeal.com/comics/printers

The Oatmeal

Comics Book NEW! Quizzes Blog Shop

Most Popular All Cats Grammar Food Animals Tech

To Make Us Miserable

By The Oatmeal http://theoatmeal.com

Terminal

```
prompt$ firefox &
[1]+ Running      firefox
prompt$ kill %1
prompt$ firefox &
[2]+ Running      firefox
[1]  Terminated   firefox
prompt$ kill %2
prompt$ kill -KILL %2
```

Terminal

```
prompt$ firefox &
[1]+ Running     firefox
prompt$ kill %1
prompt$ firefox &
[2]+ Running     firefox
[1]  Terminated  firefox
prompt$ kill %2
prompt$ kill -KILL %2
prompt$
```

- `bg` : Run a job in background mode
- `echo` : Display arguments
- `fg` : Run a job in foreground mode
- `jobs` : Display jobs
- `kill` : Signal a job
- `su` : Substitute user
- `wait` : Wait for one or more jobs
- `whoami` : Who am I

End of lecture