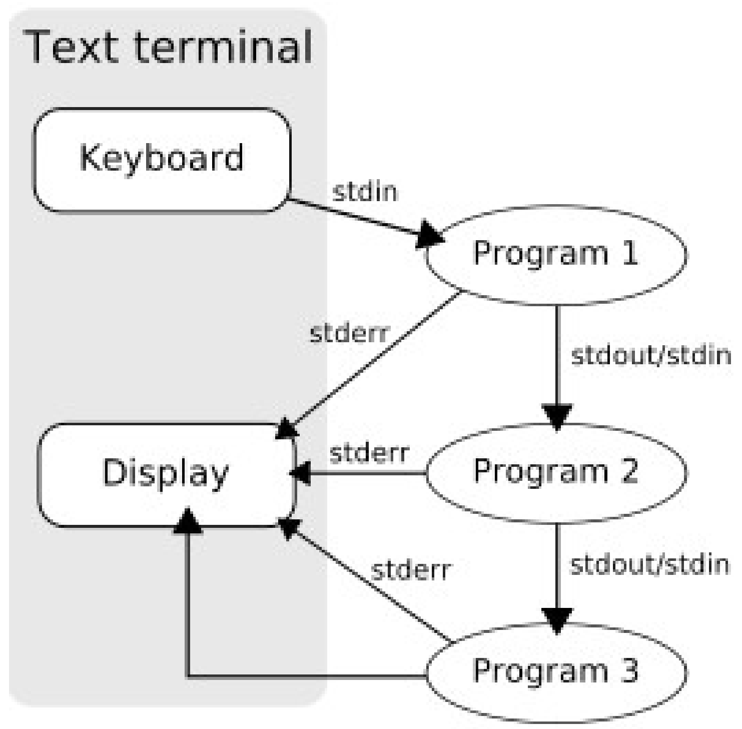
find, grep, sed, & awk

Increasing productivity with command-line tools.

Unix philosophy



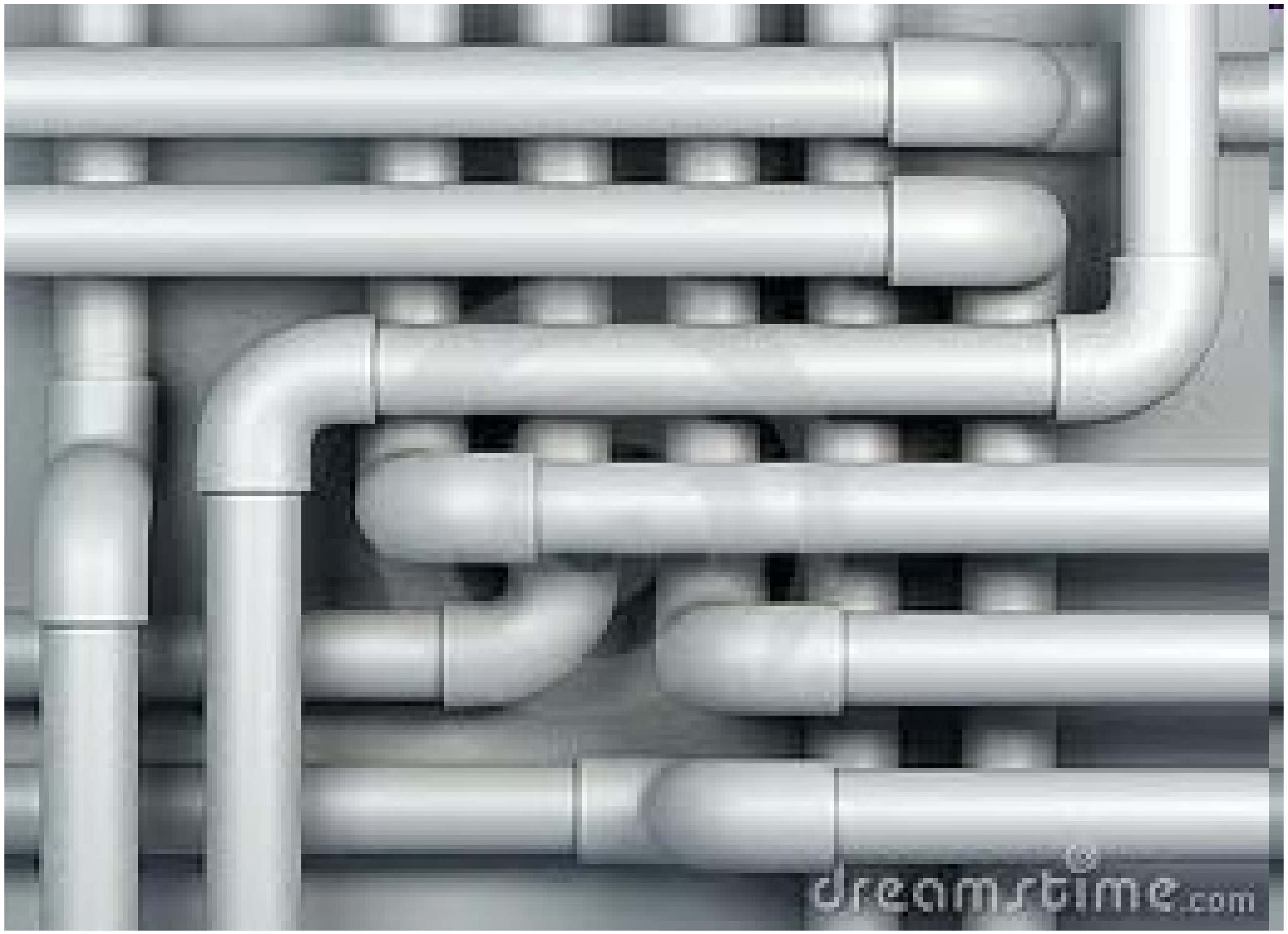
"This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface."

--Doug McIlroy, inventor of Unix pipes

Why learn command-line utils?

- Simple – “do one thing”
- Flexible – built for re-use
- Fast – no graphics, no overhead
- Ubiquitous – available on every machine
- Permanent – 40 years so far …

Part 0 – pipes and `xargs`



Some simple programs

List files in current working directory:

```
$ ls
foo bar bazoo
```

Count lines in file `foo`:

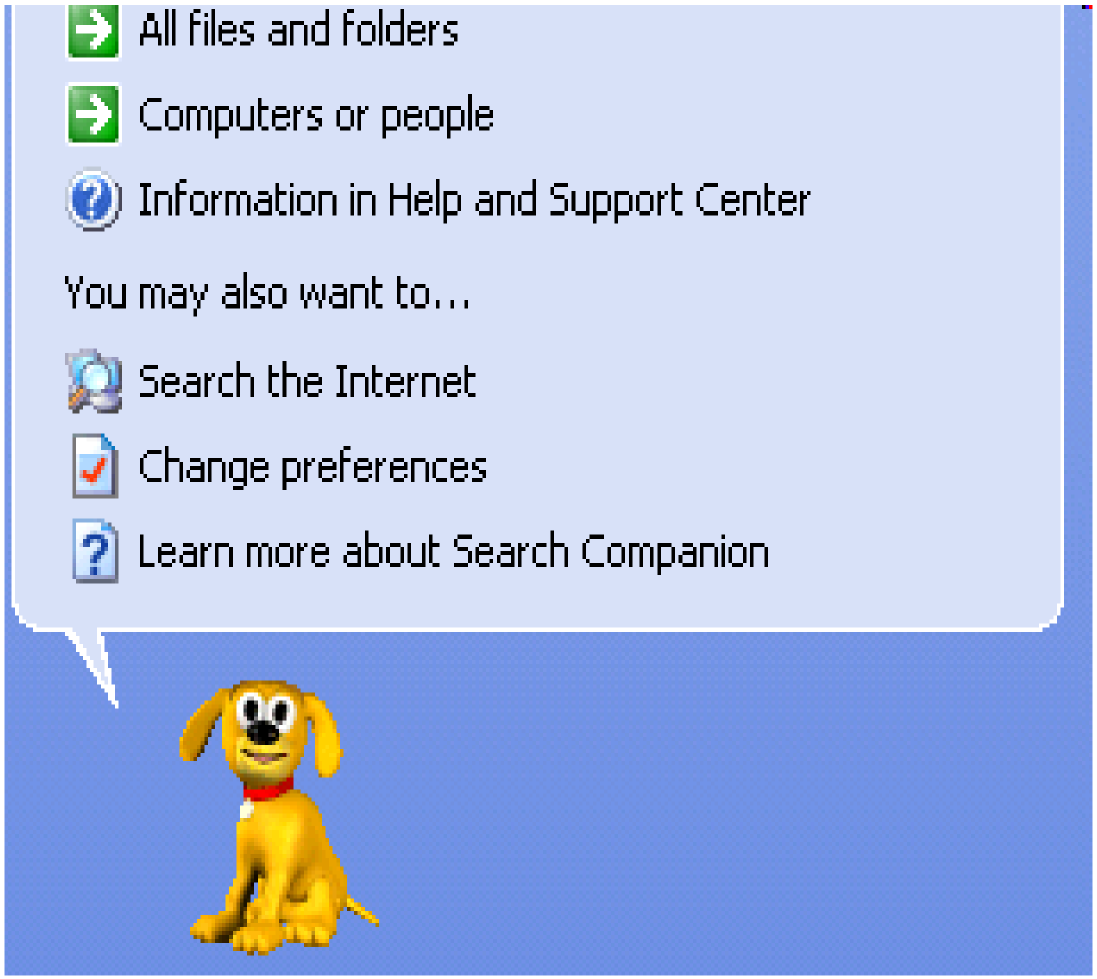
```
$ wc -l foo
42 foo
```

Putting programs together

```
$ ls | wc –l
3

$ ls | xargs wc –l
42 foo
31 bar
12 bazoo
85 total
```

Part 1: `find`


All files and folders
Computers or people
Information in Help and Support Center
You may also want to...
Search the Internet
Change preferences
Learn more about Search Companion

Basic find examples

```
$ find . –name Account.java
```

Basic find examples

```
$ find . –name Account.java
$ find /etc –name '*.conf'
```

Basic find examples

```
$ find . –name Account.java
$ find /etc –name '*.conf'
```

`$ find . –name '*.xml'`

Basic find examples

```
$ find . –name Account.java
$ find /etc –name '*.conf'
$ find . –name '*.xml'
$ find . -not -name '*.java' -maxdepth 4
```

Basic find examples

```
$ find . –name Account.java
$ find /etc –name '*.conf'
$ find . –name '*.xml'
$ find . -not -name '*.java' -maxdepth 4
$ find . \(-name '*jsp' –o –name '*xml'\)
```

Basic find examples

```
$ find . –name Account.java
$ find /etc –name '*.conf'
```
`$ find . –name '*.xml'`
```
$ find . -not -name '*.java' -maxdepth 4
$ find . \(-name '*jsp' –o –name '*xml'\)
```

- `-iname` case-insensitive
- `!` == `-not`
- `Quotes keep shell from expanding wildcards.`

Find and do stuff

```
$ find . –name '*.java' | xargs wc –l | sort
```

Find and do stuff

```
$ find . –name '*.java' | xargs wc –l | sort
```

```
Other options:
$ find . –name '*.java' -exec wc –l {} \; | sort
$ find . –name '*.java' -exec wc –l {} + | sort
```

Find and do stuff

```
$ find . –name '*.java' | xargs wc –l | sort
```

```
Other options:
$ find . –name '*.java' -exec wc –l {} \; | sort
$ find . –name '*.java' -exec wc –l {} + | sort
```

Use your imagination. `mv, rm, cp, chmod` . . .

`-exec` or `| xargs`?

- `-exec` has crazy syntax.
- `| xargs` fits Unix philosophy.
- `\;` is slow, executes command once for each line.
- `\;` not sensible, sorts 'alphabetically.'
- | xargs may fail with filenames containing whitespace, quotes or slashes.

Find by type

Files:

```
$ find . -type f
```

Find by type

Files:

```
$ find . -type f
```

Directories:

```
$ find . -type d
```

Find by type

Files:

```
$ find . -type f
```

Directories:

```
$ find . -type d
```

Links:

```
$ find . -type l
```

By modification time

Changed within day:

```
$ find . –mtime -1
```

By modification time

Changed within day:

```
$ find . –mtime -1
```

Changed within minute:

```
$ find . –mmin -15
```

By modification time

Changed within day:

```
$ find . –mtime -1
```

Changed within minute:

```
$ find . –mmin -15
```

Variants `–ctime`, `-cmin`, `-atime`, `-amin` aren't especially useful.

By modification time, `II`

Compare to file

```
$ find . -newer foo.txt
```

```
$ find . ! -newer foo.txt
```

By modification time, `III`

Compare to date

```
$ find . -type f -newermt '2010-01-01'
```

By modification time, `III`

Compare to date

```
$ find . -type f -newermt '2010-01-01'
```

Between dates!

```
$ find . -type f -newermt '2010-01-01' \
> ! -newermt '2010-06-01'
```

Find by permissions

```
$ find . –perm 644
```

```
$ find . –perm –u=w
$ find . –perm –ug=w
$ find . –perm –o=x
```

Find by size

Less than 1 kB:

```
$ find . –size -1k
```

Find by size

Less than 1 kB:

```
$ find . –size -1k
```

More than 100MB:

```
$ find . –size +100M
```

`find` summary:

- Can search by name, path, depth, permissions, type, size, modification time, and more.

`find` summary:

- Can search by name, path, depth, permissions, type, size, modification time, and more.
- Once you find what you want, pipe it to `xargs` if you want to do something with it.

`find` summary:

- Can search by name, path, depth, permissions, type, size, modification time, and more.
- Once you find what you want, pipe it to `xargs` if you want to do something with it.
- The puppy is for your grandmother.

Part 2: `grep`

global / **r**egular **e**xpression / **p**rint

From `ed` command `g/re/p`

For finding text inside files.

Basic usage:

```
$ grep <string> <file or directory>
```

Basic usage:

```
$ grep <string> <file or directory>
$ grep 'new FooDao' Bar.java
```

Basic usage:

```
$ grep <string> <file or directory>
$ grep 'new FooDao' Bar.java
$ grep Account *.xml
```

Basic usage:

```
$ grep <string> <file or directory>
$ grep 'new FooDao' Bar.java
$ grep Account *.xml
$ grep –r 'Dao[Impl|Mock]' src
```

Basic usage:

```
$ grep <string> <file or directory>
$ grep 'new FooDao' Bar.java
$ grep Account *.xml
$ grep –r 'Dao[Impl|Mock]' src
```

- Quote string if spaces or regex.
- Recursive flag is typical
- Don't quote filename with wildcards!

Common `grep` options

Case-insensitive search:

```
$ grep –i foo bar.txt
```

Common `grep` options

Case-insensitive search:

```
$ grep -i foo bar.txt
```

Only find word matches:

```
$ grep -rw foo src
```

Common `grep` options

Case-insensitive search:

```
$ grep -i foo bar.txt
```

Only find word matches:

```
$ grep -rw foo src
```

Display line number:

```
$ grep -nr 'new Foo()' src
```

Filtering results

In**v**erted search:

```
$ grep -v foo bar.txt
```

Prints lines not containing `foo`.

Filtering results

In**v**erted search:

```
$ grep -v foo bar.txt
```

Prints lines not containing `foo`.

Typical use:

```
$ grep -r User src | grep -v svn
```

Filtering results

In**v**erted search:

```
$ grep -v foo bar.txt
```

Prints lines not containing `foo`.

Typical use:

```
$ grep -r User src | grep -v svn
```

Using `find … | xargs grep …` is faster.

More `grep` options

Search for multiple terms:

```
$ grep -e foo –e bar baz.txt
```

More `grep` options

Search for multiple terms:

```
$ grep -e foo –e bar baz.txt
```

Find surrounding lines:

```
$ grep –r –C 2 foo src
```

More `grep` options

Search for multiple terms:

```
$ grep -e foo –e bar baz.txt
```

Find surrounding lines:

```
$ grep –r –C 2 foo src
```

Similarly `–A` or `–B` will print lines before and after the line containing match.

Example

Find tests that use the `AccountDao` interface.

Example

Find tests that use the `AccountDao` interface.

Possible solution (arrive at incrementally):

```
$ grep –rwn –C 3 AccountDao src/test
> | grep –v svn
```

`grep` summary:

• `-r`	**r**ecursive search
• `-i`	case **i**nsensitive
• `-w`	whole **w**ord
• `-n`	line **n**umber
• `-e`	multipl**e** searches
• `-A`	**A**fter
• `-B`	**B**efore
• `-C`	**C**entered

Part 3: sed

stream **ed**itor

For modifying files and streams of text.

sed command #1: s

```
$ echo 'foo' | sed 's/foo/bar/'
```

sed command #1: s

```
$ echo 'foo' | sed 's/foo/bar/'
bar
```

sed command #1: s

```
$ echo 'foo' | sed 's/foo/bar/'
bar

$ echo 'foo foo' | sed 's/foo/bar/'
```

sed command #1: s

```
$ echo 'foo' | sed 's/foo/bar/'
bar

$ echo 'foo foo' | sed 's/foo/bar/'
bar foo
```

sed command #1: s

```
$ echo 'foo' | sed 's/foo/bar/'
bar

$ echo 'foo foo' | sed 's/foo/bar/'
bar foo
```

`'s/foo/bar/g'` – global (within line)

Typical uses

```
$ sed 's/foo/bar/g' old
<output>
```

Typical uses

```
$ sed 's/foo/bar/g' old
<output>

$ sed 's/foo/bar/g' old > new
```

Typical uses

```
$ sed 's/foo/bar/g' old
<output>

$ sed 's/foo/bar/g' old > new

$ sed –i 's/foo/bar/g' file
```

Typical uses

```
$ sed 's/foo/bar/g' old
<output>

$ sed 's/foo/bar/g' old > new

$ sed –i 's/foo/bar/g' file

$ <stuff> | xargs sed –i 's/foo/bar/g'
```

Real life example `I`

Each time I test a batch job, a flag file gets it's only line set to `YES`, and the job can't be tested again until it is reverted to `NO`.

Real life example `I`

Each time I test a batch job, a flag file gets it's only line set to `YES`, and the job can't be tested again until it is reverted to `NO`.

```
$ sed –i 's/YES/NO/' flagfile
```

- Can change file again with up-arrow.
- No context switch.

Real life example `II`

A bunch of test cases say:

`Assert.assertStuff` which could be `assertStuff`, since using JUnit 3.

Real life example `II`

A bunch of test cases say:

`Assert.assertStuff` which could be `assertStuff`, since using JUnit 3.

```
$ find src/test/ -name '*Test.java' \
> | xargs sed –i 's/Assert.assert/assert/'
```

Real life example `III`

Windows CR-LF is mucking things up.

Real life example `III`

Windows CR-LF is mucking things up.

```
$ sed 's/.$//' winfile > unixfile
```

Replaces `\r\n` with (always inserted) `\n`

Real life example `III`

Windows CR-LF is mucking things up.

```
$ sed 's/.$//' winfile > unixfile
```

Replaces `\r\n` with (always inserted) `\n`

```
$ sed 's/$/\r/' unixfile > winfile
```

Replaces `\n` with `\r\n`.

Capturing groups

```
$ echo 'Dog Cat Pig' | sed 's/\b\(\w\)/(\1)/g'
```

Capturing groups

```
$ echo 'Dog Cat Pig' | sed 's/\b\(\w\)/(\1)/g'
(D)og (C)at (P)ig
```

Capturing groups

```
$ echo 'Dog Cat Pig' | sed 's/\b\(\w\)/(\1)/g'
(D)og (C)at (P)ig

$ echo 'john doe' | sed 's/\b\(\w\)/\U\1/g'
```

Capturing groups

```
$ echo 'Dog Cat Pig' | sed 's/\b\(\w\)/(\1)/g'
(D)og (C)at (P)ig

$ echo 'john doe' | sed 's/\b\(\w\)/\U\1/g'
John Doe
```

Capturing groups

```
$ echo 'Dog Cat Pig' | sed 's/\b\(\w\)/(\1)/g'
(D)og (C)at (P)ig

$ echo 'john doe' | sed 's/\b\(\w\)/\U\1/g'
John Doe
```

- Must escape parenthesis and braces.
- Brackets are not escaped.
- `\d` and + not supported in `sed` regex.

Exercise: formatting phone #.

Convert all strings of 10 digits to (###) ###-#####.

Exercise: formatting phone #.

Convert all strings of 10 digits to (###) ###-#####.

Conceptually, we want:

```
's/(\d{3})(\d{3})(\d{4})/(\1) \2-\3/g'
```

Exercise: formatting phone #.

Convert all strings of 10 digits to (###) ###-####.

Conceptually, we want:

```
's/(\d{3})(\d{3})(\d{4})/(\1) \2-\3/g'
```

In `sed` regex, that amounts to:

```
's/\([0-9]\{3\}\)\([0-9]\{3\}\)\([0-9]\{4\}\)/(\1) \2-\3/g'
```

Exercise: trim whitespace

Trim leading whitespace:

Exercise: trim whitespace

Trim leading whitespace:

```
$ sed -i 's/^[ \t]*//' t.txt
```

Exercise: trim whitespace

Trim leading whitespace:

```
$ sed -i 's/^[ \t]*//' t.txt
```

Trim trailing whitespace:

Exercise: trim whitespace

Trim leading whitespace:

```
$ sed -i 's/^[ \t]*//' t.txt
```

Trim trailing whitespace:

```
$ sed -i 's/[ \t]*$//' t.txt
```

Exercise: trim whitespace

Trim leading whitespace:

```
$ sed -i 's/^[ \t]*//' t.txt
```

Trim trailing whitespace:

```
$ sed -i 's/[ \t]*$//' t.txt
```

Trim leading and trailing whitespace:

Exercise: trim whitespace

Trim leading whitespace:

```
$ sed -i 's/^[ \t]*//' t.txt
```

Trim trailing whitespace:

```
$ sed –i 's/[ \t]*$//' t.txt
```

Trim leading and trailing whitespace:

```
$ sed -i 's/^[ \t]*//;s/[ \t]*$//' t.txt
```

Add comment line to file with s:

```
'1s/^/\/\/ Copyright FooCorp\n/'
```

Add comment line to file with `s`:

```
'1s/^/\/\/ Copyright FooCorp\n/'
```

- Prepends `// Copyright FooCorp\n`
- 1 restricts to first line, similar to vi search.
- ^ matches start of line.
- With `find` & `sed` insert in all `.java` files.

Shebang!

In my `.bashrc`:

```
function shebang {
  sed –i '1s/^/#!\/usr\/bin\/env python\n\n' $1
  chmod +x $1
}
```

Prepends `#!/usr/bin/env python` and makes file executable

`sed` command #2: `d`

Delete lines containing foo:

```
$ sed -i '/foo/ d' file
```

`sed` command #2: `d`

Delete lines containing foo:

```
$ sed -i '/foo/ d' file
```

Delete lines starting with `#`:

```
$ sed -i '/^#/ d' file
```

`sed` command #2: `d`

Delete lines containing foo:

```
$ sed –i '/foo/ d' file
```

Delete lines starting with `#`:

```
$ sed –i '/^#/ d' file
```

Delete first two lines:

```
$ sed –i '1,2 d' file
```

More delete examples:

Delete blank lines:

More delete examples:

Delete blank lines:

```
$ sed '/^$/ d' file
```

More delete examples:

Delete blank lines:

```
$ sed '/^$/ d' file
```

Delete up to first blank line (email header):

More delete examples:

Delete blank lines:

```
$ sed '/^$/ d' file
```

Delete up to first blank line (email header):

```
$ sed '1,/^$/ d' file
```

More delete examples:

Delete blank lines:

```
$ sed '/^$/ d' file
```

Delete up to first blank line (email header):

```
$ sed '1,/^$/ d' file
```

Note that we can combine range with regex.

Real life example `II, ctd`

A bunch of test classes have the following unnecessary line:

```
import junit.framework.Assert;
```

Real life example `II, ctd`

A bunch of test classes have the following unnecessary line:

```
import junit.framework.Assert;

$find src/test/ -name *.java | xargs \
> sed -i '/import junit.framework.Assert;/d'
```

`sed` summary

- With only `s` and `d` you should probably find a use for `sed` once a week.

`sed` summary

- With only `s` and `d` you should probably find a use for `sed` once a week.
- Combine with `find` for better results.

`sed` summary

- With only `s` and `d` you should probably find a use for `sed` once a week.
- Combine with `find` for better results.
- `sed` gets better as your regex improves.

`sed` summary

- With only `s` and `d` you should probably find a use for `sed` once a week.
- Combine with `find` for better results.
- `sed` gets better as your regex improves.
- Syntax often matches `vi`.

Part 4: awk

- **A**ho, **W**einberger, **K**ernighan
- pronounced *auk.*
- Useful for text-munging.

Simple awk programs

```
$ echo 'Jones 123' | awk '{print $0}'
Jones 123

$ echo 'Jones 123' | awk '{print $1}'
Jones

$ echo 'Jones 123' | awk '{print $2}'
123
```

Example `server.log` file:

```
fcrawler.looksmart.com [26/Apr/2000:00:00:12] "GET
/contacts.html HTTP/1.0" 200 4595 "-"
fcrawler.looksmart.com [26/Apr/2000:00:17:19] "GET
/news/news.html HTTP/1.0" 200 16716 "-"
ppp931.on.bellglobal.com [26/Apr/2000:00:16:12] "GET
/download/windows/asctab31.zip HTTP/1.0" 200 1540096
"http://www.htmlgoodies.com/downloads/freeware/webdevelopment/15.html"
123.123.123.123 [26/Apr/2000:00:23:48] "GET /pics/wpaper.gif HTTP/1.0“
200 6248 "http://www.jafsoft.com/asctortf/"
123.123.123.123 [26/Apr/2000:00:23:47] "GET /asctortf/ HTTP/1.0" 200
8130
"http://search.netscape.com/Computers/Data_Formats/Document/Text/RTF"
123.123.123.123 [26/Apr/2000:00:23:48] "GET /pics/5star2000.gif
HTTP/1.0" 200 4005 "http://www.jafsoft.com/asctortf/"
123.123.123.123 [26/Apr/2000:00:23:50] "GET /pics/5star.gif HTTP/1.0"
200 1031 "http://www.jafsoft.com/asctortf/"
123.123.123.123 [26/Apr/2000:00:23:51] "GET /pics/a2hlogo.jpg HTTP/1.0"
200 4282 "http://www.jafsoft.com/asctortf/"
<snip>
```

Built-in variables: `NF`, `NR`

- `NR` – Number of Record
- `NF` – Number of Fields
- With `$`, gives field, otherwise number

Built-in variables: NF, NR

- NR – Number of Record
- NF – Number of Fields
- With $, gives field, otherwise number

```
$ awk '{print NR, $(NF-2)}' server.log
1 200
2 200
```

Structure of an awk program

```
condition { actions }
```

Structure of an awk program

```
condition { actions }
```

```
$ awk 'END { print NR }' server.log
```

Structure of an awk program

```
condition { actions }

$ awk 'END { print NR }' server.log
9
```

Structure of an awk program

condition { actions }

```
$ awk 'END { print NR }' server.log
9

$ awk '$1 ~ /^[0-9]+.*/ { print $1,$7}' \
> server.log
```

Structure of an awk program

condition { actions }

```
$ awk 'END { print NR }' server.log
9

$ awk '$1 ~ /^[0-9]+.*/ { print $1,$7}' \
> server.log
123.123.123.123 6248
123.123.123.123 8130
```

Changing delimiter

```
$ awk 'BEGIN {FS = ":"} ; {print $2}'
```

Changing delimiter

```
$ awk 'BEGIN {FS = ":"} ; {print $2}'
```

- `FS` – Field Seperator
- `BEGIN` and `END` are special patterns

Changing delimiter

```
$ awk 'BEGIN {FS = ":"} ; {print $2}'
```

- `FS` – Field Seperator
- `BEGIN` and `END` are special patterns

Or from the command line:

```
$ awk –F: '{ print $2 }'
```

Get date out of `server.log`

```
$ awk '{ print $2 }' server.log
[26/Apr/2000:00:00:12]
```

Get date out of `server.log`

```
$ awk '{ print $2 }' server.log
[26/Apr/2000:00:00:12]

$ awk '{ print $2 }' server.log \
> | awk –F: '{print $1}
```

Get date out of `server.log`

```
$ awk '{ print $2 }' server.log
[26/Apr/2000:00:00:12]

$ awk '{ print $2 }' server.log \
> | awk –F: '{print $1}
[26/Apr/2000
```

Get date out of `server.log`

```
$ awk '{ print $2 }' server.log
[26/Apr/2000:00:00:12]

$ awk '{ print $2 }' server.log \
> | awk –F: '{print $1}
[26/Apr/2000

$ awk '{ print $2 }' server.log \
> | awk –F: '{print $1} | sed 's/\[//'
```

Get date out of `server.log`

```
$ awk '{ print $2 }' server.log
[26/Apr/2000:00:00:12]

$ awk '{ print $2 }' server.log \
> | awk –F: '{print $1}
[26/Apr/2000

$ awk '{ print $2 }' server.log \
> | awk –F: '{print $1} | sed 's/\[//'
26/Apr/2000
```

Maintaining state in `awk`

Find total bytes transferred from `server.log`

Maintaining state in awk

Find total bytes transferred from `server.log`

```
$ awk '{ b += $(NF-1) } END { print b }' server.log
1585139
```

Maintaining state in awk

Find total bytes transferred from `server.log`

```
$ awk '{ b += $(NF-1) } END { print b }' server.log
1585139
```

Find total bytes transferred to `fcrawler`

Maintaining state in awk

Find total bytes transferred from `server.log`

```
$ awk '{ b += $(NF-1) } END { print b }' server.log
1585139
```

Find total bytes transferred to `fcrawler`

```
$ awk '$1 ~ /^fcraw.*/ { b += $(NF-1) } END { print b }'\
> server.log
```

Maintaining state in awk

Find total bytes transferred from `server.log`

```
$ awk '{ b += $(NF-1) } END { print b }' server.log
1585139
```

Find total bytes transferred to `fcrawler`

```
$ awk '$1 ~ /^fcraw.*/ { b += $(NF-1) } END { print b }'\
> server.log
21311
```

One more example

Want to eliminate commented out code in large codebase.

Let's construct a one-liner to identify classes that are more than 50% comments.

One more example

Want to eliminate commented out code in large codebase.

Let's construct a one-liner to identify classes that are more than 50% comments.

```
$ awk '$1 == "//" { a+=1 } END { if (a*2 > NR)
  {print FILENAME, NR,  a}}'
```

One more example

Want to eliminate commented out code in large codebase.

Let's construct a one-liner to identify classes that are more than 50% comments.

```
$ awk '$1 == "//" { a+=1 } END { if (a*2 > NR)
  {print FILENAME, NR,  a}}'
```

To execute on all Java classes:

Example, ctd.

```
$ find src -name '*.java' -exec awk '$1 == "//"
  { a+=1 } END { if (a * 2 > NR) {print
  FILENAME, NR, a}}' {} \;
```

Example, ctd.

```
$ find src -name '*.java' -exec awk '$1 == "//"
  { a+=1 } END { if (a * 2 > NR) {print
  FILENAME, NR, a}}' {} \;
```

- Here `–exec` with `\;` is the right choice, as the `awk` program is executed for each file individually.

Example, ctd.

```
$ find src -name '*.java' -exec awk '$1 == "//"
  { a+=1 } END { if (a * 2 > NR) {print
  FILENAME, NR, a}}' {} \;
```

- Here `–exec` with `\;` is the right choice, as the `awk` program is executed for each file individually.
- It should be possible to use `xargs` and `FNR`, but I'm trying to keep the `awk` simple.

awk summary

- `NF` – Number of Field

awk summary

- `NF` – Number of Field
- `NR` – Number of Records

awk summary

- `NF` – Number of Field
- `NR` – Number of Records
- `FILENAME` – filename

awk summary

- `NF` – Number of Field
- `NR` – Number of Records
- `FILENAME` – filename
- `BEGIN, END` – special events

awk summary

- `NF` – Number of Field
- `NR` – Number of Records
- `FILENAME` – filename
- `BEGIN, END` – special events
- `FS` – Field Seperator (or `-F`).

awk summary

- `NF` – Number of Field
- `NR` – Number of Records
- `FILENAME` – filename
- `BEGIN`, `END` – special events
- `FS` – Field Seperator (or `–F`).
- `awk 'condition { actions }'`

More information

To see slides and helpful links, go to:

`http://wilsonericn.wordpress.com`

To find me at Nationwide:

`WILSOE18`

To find me on twitter:

`@wilsonericn`