CSCI 330

Shells, Part 2

Jon Lehuta

Northern Illinois University

August 17, 2020

Shells, Part 2 - Outline

Northern Illinois University

Shells, Part 2

- Shell Quoting and Escaping
- Shell Comments
- Shell Wildcards
- Regular Expressions

More `bash` shell basics

- Quoting and escaping
- Comments
- Wildcards
- Regular expressions

Northern Illinois University

Command line behavior

Some characters have special meaning for the shell

space	Separator for arguments
`$`	Variable value substitution
`=`	Variable assignment
`!`	History manipulation
`;`	Sequential execution
`` ` $() ``	Command substitution
`< > |`	I/O redirection
`&`	Background execution
`' " \`	Quoting/escaping
`#`	Comment
`* ? [] {}`	Wildcards

Quoting and Escaping

Quoting and escaping allow the use of special characters in shell commands

- Backslash (escaping)
- Single Quote
- Double Quote

Northern Illinois University

Backslash (\)

Backslash (\) is an escape character which allows special characters to be used without their special meaning:

- `\$ \" \' \< \> \{ \} \! \\`

It also allows non-special characters to represent special things.

`\a`	ring bell
`\b`	backspace
`\n`	newline
`\t`	tab
`\f`	form feed
`\u`*xxxx*	Unicode character at hex *xxxx*
`\U`*xxxxxxxx*	Unicode character at hex *xxxxxxxx*
`\U0001f4a9`	☕ ← not ice cream

Backslashes before spaces `"\ "` in the command line prevent them from separating command line arguments.

Single Quote

All characters inside `' '` preserved (except for `'`)

Shell features like variable or command substitution do not function inside of single quotes.

Examples:

```
% echo 'Joe said "Have fun"'
Joe said "Have fun"
% echo 'Joe said 'Have fun''
Joe said Have fun
```

Double Quote

Double quotes (") preserve all characters inside except for `$ ` " ! \`, which retain shell functionality.

Variables, command substitution, escaping – all allowed in double quotes.

Examples:

```
% echo "I've gone fishing"

I've gone fishing

% echo "your home directory is $HOME"

your home directory is /home/z123456
```

Northern Illinois University

Comments

In the shell, and in a lot of common scripting languages, non-quoted # character will start a single-line comment.

Everything on the rest of the line will be ignored by the shell when interpreting commands.

```
% echo This is a long line
This is a long line
% echo This is a longer line # but only part will show
This is a longer line
% # echo Anything you want, I don't care
%
```

Northern Illinois University

Wildcards: `* ? [] {}`

A pattern for matching file names on the command line, described using special characters:

`*` – allows any zero or more characters to fit in its position

```
% rm *
% ls *.txt
% wc -l assign1.*
% cp a*.txt docs
```

`?` – any single character works in this position

```
% ls assign?.cc
% wc assign?.??
% rm junk.???
```

Northern Illinois University

Wildcards: `[]` `{}`

`[..]` – current position valid if it matches any *one* of the enclosed characters

- Ex. `[a-z]` matches any *one* character in the range `a` to `z`
- `]` can be matched if it comes first – `[]x]` but not `[x]]`
- `-` can be matched if it comes first or last

If the first character after the `[` is a `!` or `^`

- then any character that is not enclosed is allowed.

`[:`*class*`:]` matches any *one* character from the *class* named, classes being:

- `alnum`, `alpha`, `blank`, `digit`, `lower`, `punct`, `upper`, etc.

`{word1,word2,word3,...}` – any *one* of the words in the `{}` can match here

Northern Illinois University

Wildcards: [] {} examples

```
% wc -l assign[123].cc
% ls csci[2-6]30
% cp [A-Z]* dir2
% rm *[!cehg]
% echo [[:upper:]]*
% cp {*.doc,*.pdf} ~
```

Regular Expressions

Regular expressions are another way of specifying patterns to match.

Unlike the wildcards above, instead of being applied to filenames by the shell, they are passed to programs, which apply them to some other string.

Composed of mostly-normal text with special meanings for characters called *meta-characters*: `.` `*` `+` `?` `[` `]` `{` `}` `(` `)`

Regular expressions are used throughout UNIX:

- Editors: `ed`, `ex`, `vi`
- Utilities: `grep`, `sed`, `awk`

There are multiple types of regular expressions:

- POSIX
  - basic
  - extended
- Perl-compatible (`pcre`)

In this course, we will be using the POSIX regular expressions.

Meta-characters

Some of the most common meta-characters are below:

meta-character	meaning
`.`	any *one* character
`[a-z]`	any *one* of listed characters
`*`	zero or more of the *preceding* atom
`?` or `\?`	either zero or one of the *preceding* atom
`+` or `\+`	one or more of the *preceding* atom

Any non-meta-character matches itself

More Meta-characters

`^`	anchor, beginning of line
`$`	anchor, end of line
`\`*char*	backslash-escape *char*
`[^`*x*`]`	any *one* character *not* in list *x*
`\<`	anchor, beginning of word
`\>`	anchor, end of word
`( )` or `\( \)`	grouping operator
`|` or `\\|`	separates alternative regular expressions
x`\{`*m*`\}`	repeat atom *x* *exactly* *m* times
x`\{`*m*`,\}`	repeat atom *x* at least *m* times
x`\{`*m*`,`*n*`\}`	repeat atom *x* between *m* and *n* times
x`\{,`*n*`\}`	repeat atom *x* at most *n* times

Basic vs Extended

- From the man page: In basic regular expressions the meta-characters `?`, `+`, `{`, `|`, `(`, and `)` lose the special meaning they have in the extended regular expressions; instead use the backslashed versions `\?`, `\+`, `\{`, `\|`, `\(`, and `\)`
- If you don't know which version you'll be using, try extended first, and if it doesn't work, add backslashes to adapt it to the basic.

`grep`

`grep` – global regular expression print

Syntax: `grep [-options] regexp [files]`

Searches for text in the files listed that matches the regular expression `regexp`

Default behavior is to show the whole line for each line that has any text that matches `regexp`. If no files are specified, it will check *standard input* for matches.

Different versions of `grep` may have a different default behavior.

- GNU `grep` defaults to extended regular expressions
- BSD `grep` defaults to basic regular expressions
- `egrep` uses extended regular expressions

Common `grep` options

`-i`	ignore case
`-w`	find only full word matches
`-A n`	show *n* lines after matches
`-B n`	show *n* lines before matches
`-C n`	show *n* lines before *and* after
`-c`	count matches
`-r`	recursively search files in directory
`-v`	invert – report lines with no matches
`-l`	show only the filename of files with matches
`-o`	show only the matches themselves
`-b`	show the position of matches on the line
`-n`	show line numbers of matches

To highlight the actual text that matched, add `--color=auto` as an option.

Northern Illinois University

`grep` Examples

```
% grep "root" /var/log/messages
% grep "r..t" /var/log/messages
% grep "bo*t" /var/log/messages
% grep "error" /var/log/*.log
```

Caveat:

Watch out for shell wildcards stealing from your regular expression if you don't use quotes, `' '` or `" "`

Regular Expressions

Formally, regular expressions consist of:

- *atoms*
- *operators*

An *atom* indicates *which* text is to be matched and *where* it is to be found.

An *operator* is used to combine regular expression atoms.

Regular Expressions - Atoms

- Any normal character (*not* a meta-character) is an atom that matches itself
- `[ ]` matches any *one* of the enclosed characters
- An *anchor* is an atom that indicates where a match must occur. It holds the regexp in place like a real anchor keeps a boat in place.
  - `^` – what comes after this must begin at the *beginning* of a *line*
  - `$` – what came before this must end at the *end* of a *line*
  - `\<` – what comes next must be at the *beginning* of a *word*
  - `\>` – what came before this must be at the *end* of a *word*
- Back references: `\1 \2 \3` ... – match things that have matched other portions of the regular expression

Northern Illinois University

`[]` Examples

`[A-H]`	`[ABCDEFGH]`	`[^AB]`	any character but `A` or `B`
`[A-Z]`	any uppercase letter	`[A-Za-z]`	any letter either case
`[0-9]`	any digit	`[^0-9]`	any character but digit
`[[a]`	`[` or `a`	`[]a]`	`]` or `a`
`[0-9\-]`	digit or `-`	`[^\^]`	anything but `^`, (not just 😄)

Short-hand classes

`[:alpha:]`	any letter of the alphabet
`[:digit:]`	any single digit
`[:alnum:]`	any letter or digit
`[:upper:]`	any uppercase letter
`[:lower:]`	any lowercase letter
`[:space:]`	any white space
`[:punct:]`	punctuation marks

Why might one want to use these?

Northern Illinois University

Anchors

Anchors tell where the next character in the pattern must be located in the text data

Anchor	Location	Example
^	Beginning of line	
$	End of line	
\<	Beginning of word	
\>	End of word	

The green arrows above show potential match positions

Back References: \n

- Used to refer to saved text in one of nine buffers
- Can refer to the text in a saved buffer by using a back reference:

For example: `\1 \2 \3 ... \9`

What text goes into each buffer is determined by what has matched the regular expression in the corresponding group. Groups are represented in the regular expression by parentheses.

Northern Illinois University

Back Reference Example

For example, the regular expression `([abc]*)x\1` would match

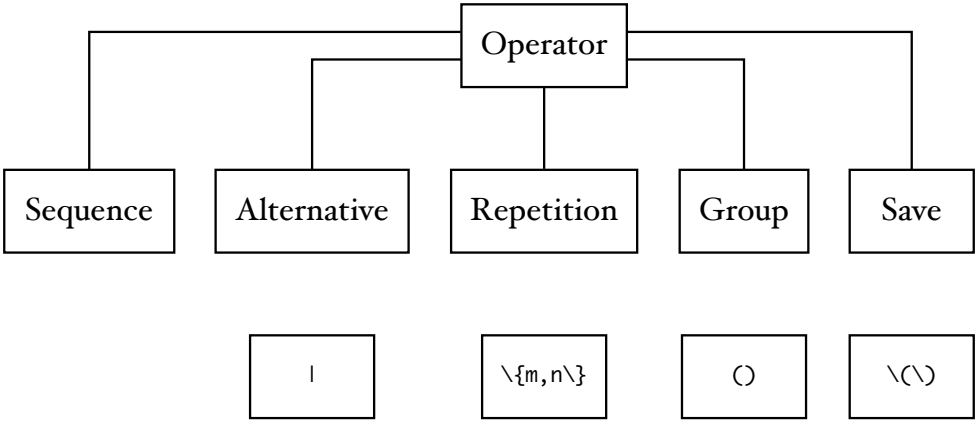
```
x       <- [abc]* matches "",    \1 also has to be empty
axa     <- [abc]* matches "a",   \1 has to match that
abxab   <- [abc]* matches "ab",  \1 has to match that
cabxcab <- [abc]* matches "cab", \1 has to match that
```

but not

```
abx
abab
cxabac
bacaca
```

Northern Illinois University

Operators



Classification of available operators for regular expressions

Sequence Operator

If you put several atoms in sequence with no operations between them, there is an implied sequence operation putting them together. The first one must appear first, then the next and the next, etc., in order to match.

`dog`	`d` then `o` then `g`
`a..b`	`a` then any two characters, then `b`
`[2-4][0-9]`	any number between 20 and 49
`[0-9][0-9]`	any two digits `00` to `99`
`^$`	matches blank lines
`^.$`	matches lines with *exactly* one character
`[0-9]-[0-9]`	any digit then a `-` then any digit

Northern Illinois University

Alternative Operator: `|` or `\|`

The operator (`|` (extended) or `\|` (basic)) is used to divide between two alternative regular expressions. The whole regular expression matches if either of the alternatives do.

`UNIX|unix`	matches `UNIX` *or* `unix`
`Mrs|Miss|Ms`	matches `Mrs`, `Miss` or `Ms`
`fe(male|mur)`	matches `female` or `femur`

Northern Illinois University

Repetition Operator: `\{ \}`

The repetition operator indicates how the atom or expression immediately preceding it may/must be repeated.

`\{`*m*`\}` – match preceding atom exactly *m* times

`\{`*m*`,`*n*`\}` – match preceding atom at least *m* times, and at most *n* times

- if either unspecified, then that side is not limited

`x\{3\}`	matches `xxx`
`A\{3,\}`	matches any string of `A`s three or longer
`A\{3,5\}`	matches `AAA`, `AAAA` or `AAAAA`
`BA\{3,5\}`	matches `BAAA`, `BAAAA` or `BAAAAA`
`(BA)\{1,3\}`	matches `BA`, `BABA`, or `BABABA`
`p\{,3\}t`	matches `t`, `pt`, `ppt`, or `pppt`

Northern Illinois University

Short Form Repetition Operators

`*`	zero or more of the preceding atom
`.*`	any zero or more characters
`+`	one or more of the preceding atom
`.+`	any one or more characters
`?`	either one or none of preceding atom
`\{0,\}`	works the same way as `*` meta-character
`\{1,\}`	works the same way as `+` meta-character
`\{0,1\}`	works the same way as `?` meta-character
`BA*`	B, BA, BAA, BAAA, BAAAA, ...
`B.*`	B, BA...BZ, BAA...BZZ, ...
`[0-9]?`	either one digit or nothing

Group Operator

In the group operator, when a group of characters is enclosed in parentheses, the next operator applies to the whole group, not only the previous characters

If you are using extended regular expressions, `( )` are sufficient.

If you are using basic regular expressions, you will need to use `\( \)` instead.