


Shell II

1. Shell II

1.1 Introduction

CSCI 330 UNIX and Network Programming



Shell II



1.2 more bash shell basics

more bash shell basics

- Wildcards
- Regular expressions
- Quoting & escaping

1.3 Command Line Behavior

Command Line Behavior

- Special characters have special meaning:

\$	variable reference	* ? + [] { }
=	assignment	• wildcards
!	event number	• regular expressions
;	command sequence	' " \
`	command substitution	• quoting & escaping
> <	i/o redirect	
&	background	

1.4 Wildcards: * ? [] { }

Wildcards: * ? [] { }

A pattern of special characters used to match file names on the command line

* zero or more characters	? exactly one character
% <code>rm *</code>	% <code>ls assign?.cc</code>
% <code>ls *.txt</code>	% <code>wc assign?..??</code>
% <code>wc -l assign1.*</code>	% <code>rm junk.???</code>
% <code>cp a*.txt docs</code>	

1.5 Wildcards: [] { }

Wildcards: [] { }

[...] matches any of the enclosed characters

[a-z] matches any character in the range a to z

- if the first character after the **[** is a **!** or **^** then any character that is not enclosed is matched
- within **[]**, **[:class:]** matches anything from a specific class: alnum, alpha, blank, digit, lower, upper, punct

{word1,word2,word3} matches any entire word

1.6 Wildcards: [] { } examples

Wildcards: [] { } examples

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

1.7 Regular Expression

Regular Expression

- Pattern of special characters to match strings
- Typically made up from special characters called meta-characters: **. * + ? [] { } ()**
- Regular expressions are used throughout UNIX:
 - utilities: grep, awk, sed, ...
- 2 types of regular expressions: basic vs. extended

1.8 Metacharacters

Metacharacters

.	Any one character, except new line
[a-z]	Any one of the enclosed characters (e.g. a-z)
*	Zero or more of preceding character
? also: \?	Zero or one of the preceding characters
+ also: \+	One or more of the preceding characters
^ or \$	Beginning or end of line
\< or \>	Beginning or end of word
() also: \(\)	Groups matched characters to be used later
also: \	Alternate
x{m,n} also: x \{m,n\}	Repetition of character x between m and n times

1.9 Basic vs. Extended

Basic vs. Extended

- Extended regular expressions use these meta-characters:

? + { } | ()

- Basic regular expressions use these meta-characters:

\? \+ \{ \} \| \ (\)

1.10 The grep Utility

The grep Utility

- searches for text in file(s)

Syntax:

grep "search-text" file(s)

- search-text is a basic regular expression

grep -E "search-text" file(s)

- search-text is an extended regular expression

1.11 The grep Utility

The grep Utility

Examples:

```
% grep "root" /var/log/auth.log
```

```
% grep "r..t" /var/log/auth.log
```

```
% grep "ro*t" /var/log/auth.log
```

```
% grep "error" /var/log/*.log
```

Caveat: watch out for shell wild cards if not using ""

1.12 Regular Expression

Regular Expression

- consists of atoms and operators
- an atom specifies what text is to be matched and where it is to be found
- an operator combines regular expression atoms

1.13 Atoms

Atoms

any character (not a meta-character) matches itself

- | | |
|--------------|--|
| . | matches any single character |
| [...] | matches any of the enclosed characters |
| ^ \$ \< \> | anchor: beginning or end of line or word |
| \1 \2 \3 ... | back reference |

1.14 Example: [...]

Example: [...]

RegExpr		Means	RegExpr		Means
[A-H]	→	[ABCDEFGH]	[^AB]	→	Any character except A or B
[A-Z]	→	Any uppercase alphabetic	[A-Za-z]	→	Any alphabetic
[0-9]	→	Any digit	[^0-9]	→	Any character except a digit
[a]	→	[or a	[a]	→] or a
[0-9\~]	→	digit or hyphen	[^\^]	→	Anything except^

1.15 short-hand classes

short-hand classes	
[alpha:]	[digit:]
• letters of the alphabet	• digits
[alnum:]	[space:]
• letters and digits	• white space
[upper:] [lower:]	[punct:]
• upper/lower case letters	• punctuation marks

1.16 Anchors

Anchors

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

Anchor		Means	Example
<code>^</code>	→	Beginning of line	One line of text.\n
<code>\$</code>	→	End of line	One line of text.\n
<code>\<</code>	→	Beginning of word	One line of text.\n
<code>\></code>	→	End of word	One line of text.\n

1.17 Back References: \n

Back References: \n

- used to retrieve saved text in one of nine buffers

ex.: `\1 \2 \3 ... \9`

- buffer defined via group operator `\(\)`

1.18 Operators

Operators

	sequence
or \	alternate
{n,m} or \{n,m\}	repetition
() or \(\)	group & save

1.19 Sequence Operator

Sequence Operator

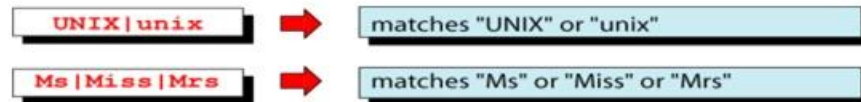
- In a sequence operator, if a series of atoms are shown in a regular expression, there is no operator between them

dog	→	matches the pattern "dog"
a..b	→	matches "a", any two characters, and "b"
[2-4][0-9]	→	matches a number between 20 and 49
[0-9][0-9]	→	matches any two digits
^\$	→	matches a blank line
^.\$	→	matches a one-character line
[0-9]-[0-9]	→	matches two digits separated by a "-"

1.20 Alternation Operator: | or \|

Alternation Operator: | or \|

- searches for one **or** more alternatives

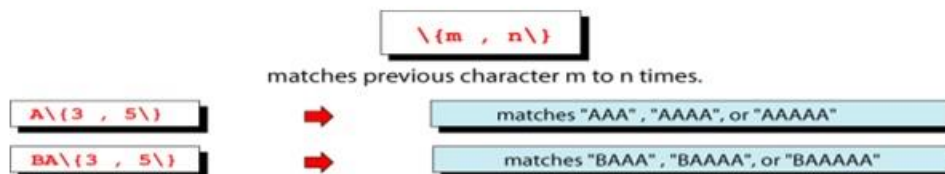


Note: Note: use \| for basic regular expression

1.21 Repetition Operator: {...} or \{...\}

Repetition Operator: {...} or \{...\}

- The repetition operator specifies that the atom or expression immediately before the repetition may be repeated



1.22 Basic Repetition Forms

Basic Repetition Forms

Formats		
<code>\{m\}</code>	→	matches previous atom exactly m times
<code>\{m, \}</code>	→	matches previous atom m times or more
<code>\{, n\}</code>	→	matches previous atom n times or less

Examples		
<code>CA\{5\}</code>	→	CAAAAA
<code>CA\{3, \}</code>	→	CAAA, CAAAA, CAAAAA, ...
<code>CA\{, 2\}</code>	→	C, CA, CAA

1.23 Short Form Repetition Operators

Short Form Repetition Operators

Formats		
<code>*</code>	→	special case: matches previous atom zero or more times
<code>+</code>	→	special case: matches previous atom one or more times
<code>?</code>	→	special case: matches previous atom 0 or one time only

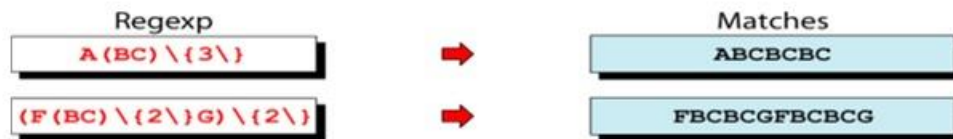
Examples		
<code>BA*</code>	→	B, BA, BAA, BAAA, BAAAA, ...
<code>B.*</code>	→	B, BA ... BZ, BAA ... BZZ, BAAA ... BZZZ, ...
<code>.*</code>	→	zero or more characters
<code>.+</code>	→	one or more characters
<code>[0-9]?</code>	→	zero or one digit

• Note: use `\+` and `\?` for basic regular expression

1.24 Group Operator: () or \ (\)

Group Operator: () or \ (\)

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



- groups define numbered buffers, can be recalled via back reference: `\1`, `\2`, `\3`, ...

1.25 Summary: Regular Expressions

Summary: Regular Expressions

.	Any one character, except new line
[a-z]	Any one of the enclosed characters (e.g. a-z)
*	Zero or more of preceding character
? also: \?	Zero or one of the preceding characters
+ also: \+	One or more of the preceding characters
^ or \$	Beginning or end of line
\< or \>	Beginning or end of word
() also: \ (\)	Groups matched characters to be used later
also: \	Alternate
x{m,n} also: x \{m,n\}	Repetition of character x between m and n times

1.26 Quoting & Escaping

Quoting & Escaping

- allows to distinguish between the literal value of a symbol and the symbols used as meta-characters
- done via the following symbols:
 - Backslash (\)
 - Single quote (')
 - Double quote (")

1.27 Backslash (\)

Backslash (\)

- also called the escape character
- preserve the character immediately following it
- For example:
to create a file named "tools>", enter:
`% touch tools\>`

1.28 Single Quote (')

Single Quote (')

- protects the literal meaning of meta-characters
 - protects all characters within pair of single quotes
- exception: it cannot protect itself

Examples:

```
% echo 'Joe said "Have fun *@!"'
Joe said "Have fun *@!"
% echo 'Joe said 'Have fun''
Joe said Have fun
```

1.29 Double Quote (")

Double Quote (")

- protects all characters within pair of double quotes, except for:
 - \$ (dollar sign) ! (event number)
 - ` (back quote) \ (backslash)

Examples:

```
% echo "I've gone fishing"
I've gone fishing
% echo "your home directory is $HOME"
your home directory is /home/student
```

1.30 Quoting Examples

Quoting Examples

```
% echo "Hello Ray []^?+*{}<>"
Hello Ray []^?+*{}<>
% echo "Hello $USER"
Hello student
% echo "It is now `date`"
It is now Mon Feb 25 10:24:08 CST 2012
% echo "you owe me \$500"
you owe me $500
```

1.31 Summary

Summary

- wildcards
- regular expressions
 - grep
- quoting