* **pwd** outputs the name of the current working directory.
* **ls** lists all files and directories in the working directory.
* **cd** switches you into the directory you specify.
* **mkdir** creates a new directory in the working directory.
* **touch** creates a new file inside the working directory.
* Options modify the behavior of commands:
  + **ls -a** lists all contents of a directory, including hidden files and directories
  + **ls -l** lists all contents in long format
  + **ls -**t orders files and directories by the time they were last modified
  + Multiple options can be used together, like ls –alt
* From the command line, you can also copy, move, and remove files and directories:
  + **cp** copies files
  + **mv** moves and renames files
  + **rm** removes files
  + **rm -r**removes directories
* The common redirection commands are:
  + **>** redirects standard output of a command to a file, overwriting previous content.
  + **>>** redirects standard output of a command to a file, appending new content to old content.
  + **<** redirects standard input to a command.
  + **|**redirects standard output of a command to another command.
* A number of other commands are powerful when combined with redirection commands:
  + **sort**: sorts lines of text alphabetically.
  + **uniq**: filters duplicate, adjacent lines of text.
  + **grep**: searches for a text pattern and outputs it.
  + **sed** : searches for a text pattern, modifies it, and outputs it.
* **export VARIABLE="Value"** sets and exports an environment variable.
* **USER** is the name of the current user.
* **PS1** is the command prompt.
* **HOME** is the home directory. It is usually not customized.
* **PATH** returns a colon separated list of file paths. It is customized in advanced cases.
* **env** returns a list of environment variables.

echo: „visszhang” amit írok azt adja vissza kimenetként.

„parancs” -help: leírása, segítsége a parancsnak

git init: git fájl-t hoz létre, git repozitori lesz ettől.

git add: hozzáad

git commit: megjelölés, állapotok

git push: github-ra teszi a dolgokat

git pull: leszedi githubról a dolgokat

**Format of the commit message**

<type>(<scope>): <subject>

<BLANK LINE>

<body>

<BLANK LINE>

<footer>

Any line of the commit message cannot be longer 100 characters! This allows the message to be easier to read on github as well as in various git tools.

### Subject line

Subject line contains succinct description of the change.

#### Allowed <type>

* feat (feature)
* fix (bug fix)
* docs (documentation)
* style (formatting, missing semi colons, …)
* refactor
* test (when adding missing tests)
* chore (maintain)

#### Allowed <scope>

Scope could be anything specifying place of the commit change. For example $location, $browser, $compile, $rootScope, ngHref, ngClick, ngView, etc...

#### <subject> text

* use imperative, present tense: “change” not “changed” nor “changes”
* don't capitalize first letter
* no dot (.) at the end

### Message body

* just as in use imperative, present tense: “change” not “changed” nor “changes”
* includes motivation for the change and contrasts with previous behavior

<http://365git.tumblr.com/post/3308646748/writing-git-commit-messages> <http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html>

### Message footer

#### Breaking changes

All breaking changes have to be mentioned in footer with the description of the change, justification and migration notes

BREAKING CHANGE: isolate scope bindings definition has changed and

the inject option for the directive controller injection was removed.

To migrate the code follow the example below:

Before:

scope: {

myAttr: 'attribute',

myBind: 'bind',

myExpression: 'expression',

myEval: 'evaluate',

myAccessor: 'accessor'

}

After:

scope: {

myAttr: '@',

myBind: '@',

myExpression: '&',

// myEval - usually not useful, but in cases where the expression is assignable, you can use '='

myAccessor: '=' // in directive's template change myAccessor() to myAccessor

}

The removed `inject` wasn't generaly useful for directives so there should be no code using it.

#### Referencing issues

Closed bugs should be listed on a separate line in the footer prefixed with "Closes" keyword like this:

Closes #234

or in case of multiple issues:

Closes #123, #245, #992

## Examples

feat($browser): onUrlChange event (popstate/hashchange/polling)

Added new event to $browser:

- forward popstate event if available

- forward hashchange event if popstate not available

- do polling when neither popstate nor hashchange available

Breaks $browser.onHashChange, which was removed (use onUrlChange instead)

fix($compile): couple of unit tests for IE9

Older IEs serialize html uppercased, but IE9 does not...

Would be better to expect case insensitive, unfortunately jasmine does

not allow to user regexps for throw expectations.

Closes #392

Breaks foo.bar api, foo.baz should be used instead

feat(directive): ng:disabled, ng:checked, ng:multiple, ng:readonly, ng:selected

New directives for proper binding these attributes in older browsers (IE).

Added coresponding description, live examples and e2e tests.

Closes #351

style($location): add couple of missing semi colons

docs(guide): updated fixed docs from Google Docs

Couple of typos fixed:

- indentation

- batchLogbatchLog -> batchLog

- start periodic checking

- missing brace

feat($compile): simplify isolate scope bindings

Changed the isolate scope binding options to:

- @attr - attribute binding (including interpolation)

- =model - by-directional model binding

- &expr - expression execution binding

This change simplifies the terminology as well as

number of choices available to the developer. It

also supports local name aliasing from the parent.

BREAKING CHANGE: isolate scope bindings definition has changed and

the inject option for the directive controller injection was removed.

To migrate the code follow the example below:

Before:

scope: {

myAttr: 'attribute',

myBind: 'bind',

myExpression: 'expression',

myEval: 'evaluate',

myAccessor: 'accessor'

}

After:

scope: {

myAttr: '@',

myBind: '@',

myExpression: '&',

// myEval - usually not useful, but in cases where the expression is assignable, you can use '='

myAccessor: '=' // in directive's template change myAccessor() to myAccessor

}

The removed `inject` wasn't generaly useful for directives so there should be no code using it.