Prelude

```
(defvar
current-user
                 (getenv
                  (if (equal system-type 'windows-nt) "USERNAME" "USER")))


(message "Prelude is powering up... Be patient, Master %s!" current-user)


(when (version< emacs-version "24.4")
 (error "Prelude requires at least GNU Emacs 24.4, but you're running %s"
emacs-version))


;; Always load newest byte code
(setq load-prefer-newer t)


(defvar prelude-dir (file-name-directory load-file-name)
 "The root dir of the Emacs Prelude distribution.")
(defvar prelude-core-dir (expand-file-name "core" prelude-dir)
 "The home of Prelude's core functionality.")
(defvar prelude-modules-dir (expand-file-name  "modules" prelude-dir)
 "This directory houses all of the built-in Prelude modules.")
(defvar prelude-personal-dir (expand-file-name "personal" prelude-dir)
 "This directory is for your personal configuration.

Users of Emacs Prelude are encouraged to keep their personal configuration
changes in this directory.   All Emacs Lisp files there are loaded
automatically
by Prelude.")
```

```
(defvar prelude-personal-preload-dir (expand-file-name "preload"
prelude-personal-dir)
 "This directory is for your personal configuration, that you want loaded
before Prelude.")
(defvar prelude-vendor-dir (expand-file-name "vendor" prelude-dir)
 "This directory houses packages that are not yet available in ELPA (or
MELPA).")
(defvar prelude-savefile-dir (expand-file-name "savefile" prelude-dir)
 "This folder stores all the automatically generated save/history-files.")
(defvar prelude-modules-file (expand-file-name "prelude-modules.el"
prelude-dir)
 "This files contains a list of modules that will be loaded by Prelude.")


(unless (file-exists-p prelude-savefile-dir)
 (make-directory prelude-savefile-dir))


(defun prelude-add-subfolders-to-load-path (parent-dir)
"Add all level PARENT-DIR subdirs to the `load-path'."
(dolist (f (directory-files parent-dir))
  (let ((name (expand-file-name f parent-dir)))
    (when (and (file-directory-p name)
               (not (string-prefix-p "." f)))
      (add-to-list 'load-path name)
      (prelude-add-subfolders-to-load-path name)))))


;; add Prelude's directories to Emacs's `load-path'
(add-to-list 'load-path prelude-core-dir)
(add-to-list 'load-path prelude-modules-dir)
(add-to-list 'load-path prelude-vendor-dir)
(prelude-add-subfolders-to-load-path prelude-vendor-dir)
```

```
;; reduce the frequency of garbage collection by making it happen on
;; each 50MB of allocated data (the default is on every 0.76MB)
(setq gc-cons-threshold 50000000)

;; warn when opening files bigger than 100MB
(setq large-file-warning-threshold 100000000)

;; preload the personal settings from `prelude-personal-preload-dir'
(when (file-exists-p prelude-personal-preload-dir)
  (message "Loading personal configuration files in %s..." prelude-personal-preload-dir)
  (mapc 'load (directory-files prelude-personal-preload-dir 't "^[^#\.].*el$")))

(message "Loading Prelude's core...")

;; the core stuff
(require 'prelude-packages)
(require 'prelude-custom)  ;; Needs to be loaded before core, editor and ui
(require 'prelude-ui)
(require 'prelude-core)
(require 'prelude-mode)
(require 'prelude-editor)
(require 'prelude-global-keybindings)

;; OSX specific settings
```

```
(when (eq system-type 'darwin)
 (require 'prelude-osx))


(message "Loading Prelude's modules...")


;; the modules
(if (file-exists-p prelude-modules-file)
   (load prelude-modules-file)
 (message "Missing modules file %s" prelude-modules-file)
 (message "You can get started by copying the bundled example file from
sample/prelude-modules.el"))


;; config changes made through the customize UI will be store here
(setq custom-file (expand-file-name "custom.el" prelude-personal-dir))


;; load the personal settings (this includes `custom-file')
(when (file-exists-p prelude-personal-dir)
   (message "Loading personal configuration files in %s..."
prelude-personal-dir)
 (mapc 'load (directory-files prelude-personal-dir 't "^[^#\.].*el$")))


(message "Prelude is ready to do thy bidding, Master %s!" current-user)


;; Patch security vulnerability in Emacs versions older than 25.3
(when (version< emacs-version "25.3")
 (eval-after-load "enriched"
   '(defun enriched-decode-display-prop (start end &optional param)
      (list start end))))
```

```
                (prelude-eval-after-init
                ;; greet the use with some useful tip
                (run-at-time 5 nil 'prelude-tip-of-the-day))


                ;;; init.el ends here
-----------------------------------


  install_p
  relude ()
  {
                printf " Cloning the Prelude's GitHub repository...\n$RESET"
                if [ x$PRELUDE_VERBOSE != x ]
                then
                    /usr/bin/env git clone $PRELUDE_URL "$PRELUDE_INSTALL_DIR"
                else
                    /usr/bin/env git clone $PRELUDE_URL "$PRELUDE_INSTALL_DIR" > /dev/null
                fi
                if ! [ $? -eq 0 ]
                then
                        printf "$RED A fatal error occurred during Prelude's installation.
              Aborting..."
                       exit 1
                fi
              }


              make_prelude_dirs () {
                printf " Making the required directories.\n$RESET"
                mkdir -p "$PRELUDE_INSTALL_DIR/savefile"
              }
```

```
colors_ () {
   case "$SHELL" in
   *zsh)
       autoload colors && colors
       eval RESET='$reset_color'
       for COLOR in RED GREEN YELLOW BLUE MAGENTA CYAN BLACK WHITE
       do
           eval $COLOR='$fg_no_bold[${(L)COLOR}]'
           eval B$COLOR='$fg_bold[${(L)COLOR}]'
       done
       ;;
   *)
       RESET='\e[0m'           # Reset
       RED='\e[0;31m'          # Red
       GREEN='\e[0;32m'        # Green
       YELLOW='\e[0;33m'       # Yellow
       BLUE='\e[0;34m'         # Blue
       PURPLE='\e[0;35m'       # Magenta
       CYAN='\e[0;36m'         # Cyan
       WHITE='\e[0;37m'        # White
       BRED='\e[1;31m'         # Bold Red
       BGREEN='\e[1;32m'       # Bold Green
       BYELLOW='\e[1;33m'      # Bold Yellow
       BBLUE='\e[1;34m'        # Bold Blue
       BPURPLE='\e[1;35m'      # Bold Magenta
       BCYAN='\e[1;36m'        # Bold Cyan
       BWHITE='\e[1;37m'       # Bold White
       ;;
   esac
```

```
}


# Commandline args:
# -d/--directory [dir]
#   Install prelude into the specified directory. If 'dir' is a relative path prefix it with $HOME.
#   Defaults to '$HOME/.emacs.d'
# -c/--colors
#   Enable colors
# -s/--source [url]
#   Clone prelude from 'url'.
#   Defaults to 'https://github.com/bbatsov/prelude.git'
# -i/--into
#   If one exists, install into the existing config
# -n/--no-bytecompile
#   Skip the compilation of the prelude files.
# -h/--help
#   Print help
# -v/--verbose
#   Verbose output, for debugging


usage() {
   printf "Usage: $0 [OPTION]\n"
   printf "  -c, --colors \t \t \t Enable colors.\n"
    printf "   -d, --directory [dir] \t Install prelude into the specified directory.\n"
   printf "  \t \t \t \t If 'dir' is a relative path prefix with $HOME.\n"
   printf "  \t \t \t \t Defaults to $HOME/.emacs.d\n"
   printf "  -s, --source [url] \t \t Clone prelude from 'url'.\n"
              printf     "          \t     \t     \t     \t     Defaults     to 'https://github.com/bbatsov/prelude.git'.\n"
```

```
    printf "   -n, --no-bytecompile \t \t Skip the bytecompilation step of prelude.\n"
    printf "   -i, --into \t \t \t Install Prelude into a subdirectory in the existing configuration\n"
    printf "   \t \t \t \t The default behavious is to install prelude into the existing\n"
    printf "   \t \t \t \t emacs configuration.\n"
    printf "   -h, --help \t \t \t Display this help and exit\n"
    printf "   -v, --verbose \t \t Display verbose information\n"
    printf "\n"
}


### Parse cli
while [ $# -gt 0 ]
do
    case $1 in
        -d | --directory)
            PRELUDE_INSTALL_DIR=$2
            shift 2
            ;;
        -c | --colors)
            colors_
            shift 1
            ;;
        -s | --source)
            PRELUDE_URL=$2
            shift 2
            ;;
        -i | --into)
            PRELUDE_INTO='true'
            shift 1
```

```
            ;;
        -n | --no-bytecompile)
            PRELUDE_SKIP_BC='true'
            shift 1
            ;;
        -h | --help)
            usage
            exit 0
            ;;
        -v | --verbose)
            PRELUDE_VERBOSE='true';
            shift 1
            ;;
        *)
            printf "Unkown option: $1\n"
            shift 1
            ;;
    esac
done

VERBOSE_COLOR=$BBLUE

[ -z "$PRELUDE_URL" ] && PRELUDE_URL="https://github.com/bbatsov/prelude.git"
[ -z "$PRELUDE_INSTALL_DIR" ] && PRELUDE_INSTALL_DIR="$HOME/.emacs.d"

if [ x$PRELUDE_VERBOSE != x ]
then
    printf "$VERBOSE_COLOR"
    printf "PRELUDE_VERBOSE = $PRELUDE_VERBOSE\n"
```

```
   printf "INSTALL_DIR = $PRELUDE_INSTALL_DIR\n"
   printf "SOURCE_URL  = $PRELUDE_URL\n"
   printf "$RESET"
   if [ -n "$PRELUDE_SKIP_BC" ]
   then
       printf "Skipping bytecompilation.\n"
   fi
   if [ -n "$PRELUDE_INTO" ]
   then
       printf "Replacing existing config (if one exists).\n"
   fi
   printf "$RESET"
fi


# If prelude is already installed
if [ -f "$PRELUDE_INSTALL_DIR/core/prelude-core.el" ]
then
   printf "\n\n$BRED"
    printf "You already have Prelude installed.$RESET\nYou'll need to remove $PRELUDE_INSTALL_DIR/prelude if you want to install Prelude again.\n"
    printf "If you want to update your copy of prelude, run 'git pull origin master' from your prelude directory\n\n"
   exit 1;
fi


### Check dependencies
printf  "$CYAN Checking to see if git is installed... $RESET"
if hash git 2>&-
then
   printf "$GREEN found.$RESET\n"
```

```
else
   printf "$RED not found. Aborting installation!$RESET\n"
   exit 1
fi;

printf  "$CYAN Checking to see if aspell is installed... "
if hash aspell 2>&-
then
   printf "$GREEN found.$RESET\n"
else
         printf   "$RED   not   found.   Install   aspell   to   benefit   from
flyspell-mode!$RESET\n"
fi

### Check emacs version
if      [      $(emacs      --version      2>/dev/null      |      sed      -n
's/.*[^0-9.]\([0-9]*\.[0-9.]*\).*/\1/p;q' | sed 's/\..*//g') -lt 24 ]
then
   printf "$YELLOW WARNING:$RESET Prelude depends on emacs $RED 24$RESET !\n"
fi

if [ -f "$HOME/.emacs" ]
then
   ## If $HOME/.emacs exists, emacs ignores prelude's init.el, so remove it
           printf    "    Backing    up    the    existing    $HOME/.emacs    to
$HOME/.emacs.pre-prelude\n"
   mv $HOME/.emacs $HOME/.emacs.pre-prelude
fi
```

```
if [ -d "$PRELUDE_INSTALL_DIR" ] || [ -f "$PRELUDE_INSTALL_DIR" ]
then
   # Existing file/directory found -> backup
            printf " Backing up the existing config to $PRELUDE_INSTALL_DIR.pre-prelude.tar.\n"
     tar -cf "$PRELUDE_INSTALL_DIR.pre-prelude.tar" "$PRELUDE_INSTALL_DIR" > /dev/null 2>&1
   PRELUDE_INSTALL_DIR_ORIG="$PRELUDE_INSTALL_DIR"
   # Overwrite existing?
                        [ -n "$PRELUDE_INTO" ] && PRELUDE_INSTALL_DIR="$PRELUDE_INSTALL_DIR/prelude"
   # Clear destination directory for git clone to work
   rm -fr "$PRELUDE_INSTALL_DIR"
   mkdir "$PRELUDE_INSTALL_DIR"
   # Replace existing config
   install_prelude
   make_prelude_dirs
   # Reinstate files that weren't replaced
      tar --skip-old-files -xf "$PRELUDE_INSTALL_DIR_ORIG.pre-prelude.tar" "$PRELUDE_INSTALL_DIR" > /dev/null 2>&1
                    [ -n "$PRELUDE_INTO" ] && cp "$PRELUDE_INSTALL_DIR/sample/prelude-modules.el" "$PRELUDE_INSTALL_DIR"
elif [ -e "$PRELUDE_INSTALL_DIR" ]
then
   # File exist but not a regular file or directory
   # WTF NOW?
   printf "$BRED $PRELUDE_INSTALL_DIR exist but isn't a file or directory.\n"
    printf "$BRED please remove this file or install prelude in a different directory"
   printf "$BRED (-d flag)\n$RESET"
   exit 1
else
   # Nothing yet so just install prelude
```

```
    install_prelude
    make_prelude_dirs
    cp "$PRELUDE_INSTALL_DIR/sample/prelude-modules.el" "$PRELUDE_INSTALL_DIR"
fi


if [ -z "$PRELUDE_SKIP_BC" ];
then
    if which emacs > /dev/null 2>&1
    then
        printf " Bytecompiling Prelude.\n"
        if [ x$PRELUDE_VERBOSE != x ]
        then
            emacs -batch -f batch-byte-compile "$PRELUDE_INSTALL_DIR/core"/*.el
        else
             emacs -batch -f batch-byte-compile "$PRELUDE_INSTALL_DIR/core"/*.el
> /dev/null 2>&1
        fi
    else
        printf "$YELLOW Emacs not found.$RESET Skipping bytecompilation.\n"
    fi
else
    printf "Skipping bytecompilation.\n"
fi


printf "\n"
printf "$BBLUE  ____            _           _       \n"
printf "$BBLUE |  _ \ _ __ ___| |_   _  __| | ___  \n"
printf "$BBLUE | |_) |  __/ _ \ | | | |/ _  |/ _ \ \n"
printf "$BBLUE |  __/| | |  __/ | |_| | (_| |  __/ \n"
printf "$BBLUE |_|   |_|  \___|_|\__,_|\__,_|\___| \n\n"
```

```
        printf "$GREEN ... is now installed and ready to do thy bidding, Master $USER!$RESET\n"


;;; Uncomment the modules you'd like to use and restart Prelude afterwards

;; Emacs IRC client
(require 'prelude-erc)
(require 'prelude-ido) ;; Super charges Emacs completion for C-x C-f and more
;; (require 'prelude-helm) ;; Interface for narrowing and search
;; (require 'prelude-helm-everywhere) ;; Enable Helm everywhere
(require 'prelude-company)
;; (require 'prelude-key-chord) ;; Binds useful features to key combinations
;; (require 'prelude-mediawiki)
;; (require 'prelude-evil)


;;; Programming languages support
(require 'prelude-c)
;; (require 'prelude-clojure)
;; (require 'prelude-coffee)
;; (require 'prelude-common-lisp)
;; (require 'prelude-css)
(require 'prelude-emacs-lisp)
;; (require 'prelude-erlang)
;; (require 'prelude-elixir)
;; (require 'prelude-go)
;; (require 'prelude-haskell)
```

```
(require 'prelude-js)

;; (require 'prelude-latex)

(require 'prelude-lisp)

;; (require 'prelude-ocaml)

(require 'prelude-org) ;; Org-mode helps you keep TODO lists, notes and more
(require 'prelude-perl)

;; (require 'prelude-python)

;; (require 'prelude-ruby)

;; (require 'prelude-rust)

;; (require 'prelude-scala)

(require 'prelude-scheme)

(require 'prelude-shell)

;; (require 'prelude-scss)

;; (require 'prelude-web) ;; Emacs mode for web templates
(require 'prelude-xml)

;; (require 'prelude-yaml)
```

```
;;; Code:

(require 'prelude-programming)

(defun prelude-c-mode-common-defaults ()
  (setq c-default-style "k&r"
        c-basic-offset 4)
  (c-set-offset 'substatement-open 0))

(setq prelude-c-mode-common-hook 'prelude-c-mode-common-defaults)

;; this will affect all modes derived from cc-mode, like
;; java-mode, php-mode, etc
(add-hook 'c-mode-common-hook (lambda ()
                                (run-hooks 'prelude-c-mode-common-hook)))

(defun prelude-makefile-mode-defaults ()
  (whitespace-toggle-options '(tabs))
  (setq indent-tabs-mode t ))

(setq prelude-makefile-mode-hook 'prelude-makefile-mode-defaults)

(add-hook 'makefile-mode-hook (lambda ()
                                (run-hooks 'prelude-makefile-mode-hook)))
(provide 'prelude-c)

;;; prelude-c.el ends here
```

```
prelude-require-packages '(evil goto-chg
evil-surround evil-visualstar evil-numbers))

                                                    (require 'evil-visualstar)


                                                    (setq evil-mode-line-format 'before)
```

```
(setq evil-emacs-state-cursor  '("red"
box))
(setq evil-normal-state-cursor '("gray"
box))
(setq evil-visual-state-cursor '("gray"
box))
(setq evil-insert-state-cursor '("gray"
bar))
(setq evil-motion-state-cursor '("gray"
box))


;; prevent esc-key from translating to
meta-key in terminal mode
(setq evil-esc-delay 0)


(evil-mode 1)

(global-evil-surround-mode 1)


(define-key evil-normal-state-map (kbd
"C-A")
 'evil-numbers/inc-at-pt)

(define-key evil-normal-state-map (kbd
"C-S-A")
 'evil-numbers/dec-at-pt)


;;

;; Other useful Commands

;;

(evil-ex-define-cmd "W"
'evil-write-all)
(evil-ex-define-cmd "Tree"
'speedbar-get-focus)
(evil-ex-define-cmd "linum" 'linum-mode)
```

```
(evil-ex-define-cmd "Align"
'align-regexp)


(defun prelude-yank-to-end-of-line ()
 "Yank to end of line."
 (interactive)
 (evil-yank (point) (point-at-eol)))


(define-key evil-normal-state-map
 (kbd "Y") 'prelude-yank-to-end-of-line)


(defun prelude-shift-left-visual ()
 "Shift left and restore visual
selection."
 (interactive)
 (evil-shift-left (region-beginning)
(region-end))
 (evil-normal-state)
 (evil-visual-restore))


(defun prelude-shift-right-visual ()
 "Shift right and restore visual
selection."
 (interactive)
 (evil-shift-right (region-beginning)
(region-end))
 (evil-normal-state)
 (evil-visual-restore))
```

```
(define-key evil-visual-state-map (kbd
">") 'prelude-shift-right-visual)
(define-key evil-visual-state-map (kbd
"<") 'prelude-shift-left-visual)


;; Scrolling

(defun
prelude-evil-scroll-down-other-window ()
 (interactive)

 (scroll-other-window))



(defun
prelude-evil-scroll-up-other-window ()
 (interactive)

 (scroll-other-window '-))



(define-key evil-normal-state-map

 (kbd "C-S-d")
'prelude-evil-scroll-down-other-window)



(define-key evil-normal-state-map

 (kbd "C-S-u")
'prelude-evil-scroll-up-other-window)



;;

;; Magit from avsej

;;

(evil-add-hjkl-bindings
magit-log-mode-map 'emacs)
(evil-add-hjkl-bindings
magit-commit-mode-map 'emacs)
```

```
(evil-add-hjkl-bindings
magit-branch-manager-mode-map 'emacs
 "K" 'magit-discard
 "L" 'magit-log-popup)
(evil-add-hjkl-bindings
magit-status-mode-map 'emacs
 "K" 'magit-discard
 "l" 'magit-log-popup
 "h" 'magit-diff-toggle-refine-hunk)


(setq evil-shift-width 2)


;;; enable avy with evil-mode
(define-key evil-normal-state-map (kbd
"SPC") 'avy-goto-word-1)


;;; snagged from Eric S. Fraga
;;;
http://lists.gnu.org/archive/html/emacs-
orgmode/2012-05/msg00153.html
(defun prelude-evil-key-bindings-for-org
()
 ;;(message "Defining evil key bindings
for org")
 (evil-declare-key 'normal org-mode-map
   "gk" 'outline-up-heading
   "gj" 'outline-next-visible-heading
   "H" 'org-beginning-of-line ; smarter
behaviour on headlines etc.
   "L" 'org-end-of-line ; smarter
behaviour on headlines etc.
   "t" 'org-todo ; mark a TODO item as
DONE
```

```
    ",c" 'org-cycle
    (kbd "TAB") 'org-cycle
    ",e" 'org-export-dispatch
    ",n" 'outline-next-visible-heading
    ",p" 'outline-previous-visible-heading
    ",t" 'org-set-tags-command
    ",u" 'outline-up-heading
    "$" 'org-end-of-line ; smarter behaviour on headlines etc.
    "^" 'org-beginning-of-line ; ditto
    "-" 'org-ctrl-c-minus ; change bullet style
    "<" 'org-metaleft ; out-dent
    ">" 'org-metaright ; indent
    ))
(prelude-evil-key-bindings-for-org)
(provide 'prelude-evil)
```

;;;
Code:

```
(require 'prelude-programming)

(prelude-require-packages '(go-mode
                            company-go
                            go-eldoc
                            go-projectile
```

```
                            gotest))


(require 'go-projectile)


;; Ignore go test -c output files
(add-to-list 'completion-ignored-extensions ".test")


(define-key 'help-command (kbd "G") 'godoc)


(eval-after-load 'go-mode
 '(progn
    (defun prelude-go-mode-defaults ()
      ;; Add to default go-mode key bindings
      (let ((map go-mode-map))
           (define-key map (kbd "C-c a") 'go-test-current-project) ;; current
package, really
        (define-key map (kbd "C-c m") 'go-test-current-file)
        (define-key map (kbd "C-c .") 'go-test-current-test)
        (define-key map (kbd "C-c b") 'go-run)
        (define-key map (kbd "C-h f") 'godoc-at-point))


      ;; Prefer goimports to gofmt if installed
      (let ((goimports (executable-find "goimports")))
        (when goimports
          (setq gofmt-command goimports)))


      ;; gofmt on save
```

```
        (add-hook 'before-save-hook 'gofmt-before-save nil t)

        ;; stop whitespace being highlighted
        (whitespace-toggle-options '(tabs))

        ;; Company mode settings
        (set (make-local-variable 'company-backends) '(company-go))

        ;; El-doc for Go
        (go-eldoc-setup)

        ;; CamelCase aware editing operations
        (subword-mode +1))

    (setq prelude-go-mode-hook 'prelude-go-mode-defaults)

    (add-hook 'go-mode-hook (lambda ()
                              (run-hooks 'prelude-go-mode-hook)))))

(provide 'prelude-go)
;;; prelude-go.el ends here
```

;;;
Code:

```
(require 'prelude-helm)

(prelude-require-packages '(helm-descbinds helm-ag))

(require 'helm-eshell)



(global-set-key (kbd "M-x") 'helm-M-x)

(global-set-key (kbd "C-x C-m") 'helm-M-x)

(global-set-key (kbd "M-y") 'helm-show-kill-ring)

(global-set-key (kbd "C-x b") 'helm-mini)

(global-set-key (kbd "C-x C-b") 'helm-buffers-list)

(global-set-key (kbd "C-x C-f") 'helm-find-files)

(global-set-key (kbd "C-h f") 'helm-apropos)

(global-set-key (kbd "C-h r") 'helm-info-emacs)

(global-set-key (kbd "C-h C-l") 'helm-locate-library)

(define-key prelude-mode-map (kbd "C-c f") 'helm-recentf)



(define-key minibuffer-local-map (kbd "C-c C-l") 'helm-minibuffer-history)



(define-key isearch-mode-map (kbd "C-o") 'helm-occur-from-isearch)



;; shell history.

(define-key shell-mode-map (kbd "C-c C-l") 'helm-comint-input-ring)



;; use helm to list eshell history

(add-hook 'eshell-mode-hook

          #'(lambda ()
```

```
                (substitute-key-definition 'eshell-list-history 'helm-eshell-history
        eshell-mode-map)))


        (substitute-key-definition 'find-tag 'helm-etags-select global-map)

        (setq projectile-completion-system 'helm)

        (helm-descbinds-mode)

        (helm-mode 1)


        ;; enable Helm version of Projectile with replacment commands

        (helm-projectile-on)


        (provide 'prelude-helm-everywhere)
```

```
(prelude-require-packages     '(helm
helm-projectile))


                                    (require 'helm-config)

                                    (require 'helm-projectile)


                                    (when (executable-find "curl")

                                     (setq helm-google-suggest-use-curl-p t))


                                    ;;  See  https://github.com/bbatsov/prelude/pull/670
                                    for a detailed
                                    ;; discussion of these options.
```

```
(setq helm-split-window-in-side-p           t
      helm-buffers-fuzzy-matching           t
      helm-move-to-line-cycle-in-source     t
      helm-ff-search-library-in-sexp        t
      helm-ff-file-name-history-use-recentf t)


;; The default "C-x c" is quite close to "C-x C-c",
which quits Emacs.
;; Changed to "C-c h". Note: We must set "C-c h"
globally, because we
;; cannot change `helm-command-prefix-key' once
`helm-config' is loaded.
(global-set-key (kbd "C-c h") 'helm-command-prefix)
(global-unset-key (kbd "C-x c"))


(define-key helm-command-map (kbd "o")
'helm-occur)
(define-key helm-command-map (kbd "g")
'helm-do-grep)
(define-key helm-command-map (kbd "C-c w")
'helm-wikipedia-suggest)
(define-key helm-command-map (kbd "SPC")
'helm-all-mark-rings)


(push "Press <C-c p h> to navigate a project in
Helm." prelude-tips)


(provide 'prelude-helm)


;;; prelude-helm.el ends here
```

;;;
Code:

```
(prelude-require-packages '(flx-ido ido-completing-read+ smex))


(require 'ido)
(require 'ido-completing-read+)
(require 'flx-ido)


(setq ido-enable-prefix nil
      ido-enable-flex-matching t
      ido-create-new-buffer 'always
      ido-use-filename-at-point 'guess
      ido-max-prospects 10
                  ido-save-directory-list-file    (expand-file-name    "ido.hist"
prelude-savefile-dir)
      ido-default-file-method 'selected-window
      ido-auto-merge-work-directories-length -1)
(ido-mode +1)
(ido-ubiquitous-mode +1)


;;; smarter fuzzy matching for ido
(flx-ido-mode +1)
;; disable ido faces to see flx highlights
(setq ido-use-faces nil)


;;; smex, remember recently and most frequently used commands
(require 'smex)
```

```
(setq smex-save-file (expand-file-name ".smex-items" prelude-savefile-dir))
(smex-initialize)
(global-set-key (kbd "M-x") 'smex)
(global-set-key (kbd "M-X") 'smex-major-mode-commands)


(provide 'prelude-ido)
;;; prelude-ido.el ends here
```

;;;
Code:

```
(prelude-require-packages '(web-mode))


(require 'web-mode)


(add-to-list 'auto-mode-alist '("\\.phtml\\'" . web-mode))
(add-to-list 'auto-mode-alist '("\\.tpl\\.php\\'" . web-mode))
(add-to-list 'auto-mode-alist '("\\.tpl\\'" . web-mode))
(add-to-list 'auto-mode-alist '("\\.blade\\.php\\'" . web-mode))
(add-to-list 'auto-mode-alist '("\\.jsp\\'" . web-mode))
(add-to-list 'auto-mode-alist '("\\.as[cp]x\\'" . web-mode))
(add-to-list 'auto-mode-alist '("\\.erb\\'" . web-mode))
(add-to-list 'auto-mode-alist '("\\.html?\\'" . web-mode))
(add-to-list 'auto-mode-alist
'("/\\(views\\|html\\|theme\\|templates\\)/.*\\.php\\'" . web-mode))
```

```
;; make web-mode play nice with smartparens
(setq web-mode-enable-auto-pairing nil)


(sp-with-modes '(web-mode)
 (sp-local-pair "%" "%"
                :unless '(sp-in-string-p)
                :post-handlers '(((lambda (&rest _ignored)
                                    (just-one-space)
                                    (save-excursion (insert " ")))
                                  "SPC" "=" "#")))
 (sp-local-tag "%" "<% "  " %>")
 (sp-local-tag "=" "<%= " " %>")
 (sp-local-tag "#" "<%# " " %>"))


(eval-after-load 'web-mode
 '(progn
    (defun prelude-web-mode-defaults ())
    (setq prelude-web-mode-hook 'prelude-web-mode-defaults)


    (add-hook 'web-mode-hook (lambda ()
                               (run-hooks 'prelude-web-mode-hook)))))


(provide 'prelude-web)
;;; prelude-web.el ends here
```

```
(require
'prelude-programming)
                        (prelude-require-packages '(rainbow-delimiters))


                        ;; Lisp configuration
                        (define-key read-expression-map (kbd "TAB") 'completion-at-point)


                        ;; wrap keybindings
                        (define-key lisp-mode-shared-map (kbd "M-(") (prelude-wrap-with
                        "("))
                        ;; FIXME: Pick terminal-friendly binding.
                        ;;(define-key lisp-mode-shared-map (kbd "M-[") (prelude-wrap-with
                        "["))
                        (define-key lisp-mode-shared-map (kbd "M-\"") (prelude-wrap-with
                        "\""))


                        ;; a great lisp coding hook
                        (defun prelude-lisp-coding-defaults ()
                         (smartparens-strict-mode +1)
                         (rainbow-delimiters-mode +1))


                        (setq prelude-lisp-coding-hook 'prelude-lisp-coding-defaults)


                        ;; interactive modes don't need whitespace checks
                        (defun prelude-interactive-lisp-coding-defaults ()
                         (smartparens-strict-mode +1)
                         (rainbow-delimiters-mode +1)
                         (whitespace-mode -1))
```

```
                        (setq                        prelude-interactive-lisp-coding-hook
                        'prelude-interactive-lisp-coding-defaults)

                        (provide 'prelude-lisp)

                        ;;; prelude-lisp.el ends here
```

```
  ;; Boston, MA 02110-1301,
  USA.

                           ;;; Code:

                           (add-to-list 'auto-mode-alist '("\\.org\\'" . org-mode))
                           (global-set-key "\C-cl" 'org-store-link)
                           (global-set-key "\C-ca" 'org-agenda)
                           (global-set-key "\C-cb" 'org-iswitchb)
                           (setq org-log-done t)

                           (defun prelude-org-mode-defaults ()
                                (let      ((oldmap      (cdr      (assoc      'prelude-mode
                           minor-mode-map-alist)))
                                 (newmap (make-sparse-keymap)))
                             (set-keymap-parent newmap oldmap)
                             (define-key newmap (kbd "C-c +") nil)
```

```
   (define-key newmap (kbd "C-c -") nil)
   (define-key newmap (kbd "C-a") nil)
   (make-local-variable 'minor-mode-overriding-map-alist)
                (push `(prelude-mode . ,newmap)
minor-mode-overriding-map-alist))
)

(setq prelude-org-mode-hook 'prelude-org-mode-defaults)

(add-hook 'org-mode-hook (lambda () (run-hooks
'prelude-org-mode-hook)))

(provide 'prelude-org)
```

--