```
**********************************************************
    1409 Thu Nov  6 11:54:31 2014
new/usr/src/tools/findunref/Makefile
5292 findunref is both slow and broken
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #

  22 #
  23 # Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
  24 # Use is subject to license terms.
  25 #

  27 PROG      = findunref
  28 MAN1FILES = findunref.1
  29 CFLAGS += $(CCVERBOSE)
  30 LINTFLAGS += -ux

  32 include ../Makefile.tools

  34 CERRWARN += -_gcc=-Wno-unused
  35 CERRWARN += -_gcc=-Wno-parentheses

  34 $(ROOTONBLDMAN1FILES) := FILEMODE=      644

  36 EXCEPTION_SRC= common open
  37 EXCEPTION_LISTS=        $(EXCEPTION_SRC:%=exception_list.%)

  39 CLOBBERFILES += exception_list

  41 .KEEP_STATE:

  43 all:    $(PROG) exception_list

  45 install: all .WAIT $(ROOTONBLDMACHPROG) $(ROOTONBLDMAN1FILES)

  47 lint:   lint_PROG

  49 exception_list: $(EXCEPTION_LISTS)
  50         -$(RM) $@
  51         $(CAT) $(EXCEPTION_LISTS) > $@

  53 clean:

  55 include ../Makefile.targ
```

```
**********************************************************
   13317 Thu Nov  6 11:54:32 2014
new/usr/src/tools/findunref/findunref.c
5292 findunref is both slow and broken
**********************************************************
______unchanged_portion_omitted_

  55 /*
  56  * Data associated with the current SCM manifest.
  56  * Data associated with the current Mercurial manifest.
  57  */
  58 typedef struct scmdata {
  58 typedef struct hgdata {
  59         pnset_t         *manifest;
  60         char            metapath[MAXPATHLEN];
  60         char            hgpath[MAXPATHLEN];
  61         char            root[MAXPATHLEN];
  62         unsigned int    rootlen;
  63         boolean_t       rootwarn;
  64 } scmdata_t;
  64 } hgdata_t;

  66 /*
  67  * Hooks used to check if a given unreferenced file is known to an SCM
  68  * (currently Git, Mercurial and TeamWare).
  68  * (currently Mercurial and TeamWare).
  69  */
  70 typedef int checkscm_func_t(const char *, const struct FTW *);
  71 typedef void chdirscm_func_t(const char *);

  73 typedef struct {
  74         const char      *name;
  75         checkscm_func_t *checkfunc;
  76         chdirscm_func_t *chdirfunc;
  77 } scm_t;

  79 static checkscm_func_t check_tw, check_scmdata;
  79 static checkscm_func_t check_tw, check_hg, check_git;
  80 static chdirscm_func_t chdir_hg, chdir_git;
  81 static int      pnset_add(pnset_t *, const char *);
  82 static int      pnset_check(const pnset_t *, const char *);
  83 static void     pnset_empty(pnset_t *);
  84 static void     pnset_free(pnset_t *);
  85 static int      checkpath(const char *, const struct stat *, int, struct FTW *);
  86 static pnset_t  *make_exset(const char *);
  87 static void     warn(const char *, ...);
  88 static void     die(const char *, ...);

  90 static const scm_t scms[] = {
  91         { "tw",         check_tw,       NULL            },
  92         { "teamware",   check_tw,       NULL            },
  93         { "hg",         check_scmdata,  chdir_hg        },
  94         { "mercurial",  check_scmdata,  chdir_hg        },
  95         { "git",        check_scmdata,  chdir_git       },
  93         { "hg",         check_hg,       chdir_hg        },
  94         { "mercurial",  check_hg,       chdir_hg        },
  95         { "git",        check_git,      chdir_git       },
  96         { NULL,         NULL,           NULL            }
  97 };

  99 static const scm_t      *scm;
 100 static scmdata_t        scmdata;
 100 static hgdata_t         hgdata;
 101 static pnset_t          *gitmanifest = NULL;
 101 static time_t           tstamp;         /* timestamp to compare files to */
 102 static pnset_t          *exsetp;        /* pathname globs to ignore */
```



```
 103 static const char       *progname;

 105 int
 106 main(int argc, char *argv[])
 107 {
 108         int c;
 109         char path[MAXPATHLEN];
 110         char subtree[MAXPATHLEN] = "./";
 111         char *tstampfile = ".build.tstamp";
 112         struct stat tsstat;

 114         progname = strrchr(argv[0], '/');
 115         if (progname == NULL)
 116                 progname = argv[0];
 117         else
 118                 progname++;

 120         while ((c = getopt(argc, argv, "as:t:S:")) != EOF) {
 121                 switch (c) {
 122                 case 'a':
 123                         /* for compatibility; now the default */
 124                         break;

 126                 case 's':
 127                         (void) strlcat(subtree, optarg, MAXPATHLEN);
 128                         break;

 130                 case 't':
 131                         tstampfile = optarg;
 132                         break;

 134                 case 'S':
 135                         for (scm = scms; scm->name != NULL; scm++) {
 136                                 if (strcmp(scm->name, optarg) == 0)
 137                                         break;
 138                         }
 139                         if (scm->name == NULL)
 140                                 die("unsupported SCM '%s'\n", optarg);
 141                         break;

 143                 default:
 144                 case '?':
 145                         goto usage;
 146                 }
 147         }

 149         argc -= optind;
 150         argv += optind;

 152         if (argc != 2) {
 153 usage:          (void) fprintf(stderr, "usage: %s [-s <subtree>] "
 154                     "[-t <tstampfile>] [-S hg|tw|git] <srcroot> <exceptfile>\n",
 155                     progname);
 156                 return (EXIT_FAILURE);
 157         }

 159         /*
 160          * Interpret a relative timestamp path as relative to srcroot.
 161          */
 162         if (tstampfile[0] == '/')
 163                 (void) strlcpy(path, tstampfile, MAXPATHLEN);
 164         else
 165                 (void) snprintf(path, MAXPATHLEN, "%s/%s", argv[0], tstampfile);

 167         if (stat(path, &tsstat) == -1)
 168                 die("cannot stat timestamp file \"%s\"", path);
```

```
169         tstamp = tsstat.st_mtime;

171         /*
172          * Create the exception pathname set.
173          */
174         exsetp = make_exset(argv[1]);
175         if (exsetp == NULL)
176                 die("cannot make exception pathname set\n");

178         /*
179          * Walk the specified subtree of the tree rooted at argv[0].
180          */
181         if (chdir(argv[0]) == -1)
182                 die("cannot change directory to \"%s\"", argv[0]);

184         if (nftw(subtree, checkpath, 100, FTW_PHYS) != 0)
185                 die("cannot walk tree rooted at \"%s\"\n", argv[0]);

187         pnset_empty(exsetp);
188         return (EXIT_SUCCESS);
189 }

191 /*
192  * Load and return a pnset for the manifest for the Mercurial repo at ‘hgroot’.
193  */
194 static pnset_t *
195 hg_manifest(const char *hgroot)
196 load_manifest(const char *hgroot)
196 {
197         FILE    *fp = NULL;
198         char    *hgcmd = NULL;
199         char    *newline;
200         pnset_t *pnsetp;
201         char    path[MAXPATHLEN];

203         pnsetp = calloc(sizeof (pnset_t), 1);
204         if (pnsetp == NULL ||
205             asprintf(&hgcmd, "hg manifest -R %s", hgroot) == -1)
206                 goto fail;

208         fp = popen(hgcmd, "r");
209         if (fp == NULL)
210                 goto fail;

212         while (fgets(path, sizeof (path), fp) != NULL) {
213                 newline = strrchr(path, '\n');
214                 if (newline != NULL)
215                         *newline = '\0';

217                 if (pnset_add(pnsetp, path) == 0)
218                         goto fail;
219         }

221         (void) pclose(fp);
222         free(hgcmd);
223         return (pnsetp);
224 fail:
225         warn("cannot load hg manifest at %s", hgroot);
226         if (fp != NULL)
227                 (void) pclose(fp);
228         free(hgcmd);
229         pnset_free(pnsetp);
230         return (NULL);
231 }

233 /*
```



```
234  * Load and return a pnset for the manifest for the Git repo at ‘gitroot’.
235  */
236 static pnset_t *
237 git_manifest(const char *gitroot)
234 static void
235 chdir_git(const char *path)
238 {
239         FILE    *fp = NULL;
240         char    *gitcmd = NULL;
241         char    *newline;
240         char fn[MAXPATHLEN];
242         pnset_t *pnsetp;
243         char    path[MAXPATHLEN];
244 #endif /* ! codereview */

246         pnsetp = calloc(sizeof (pnset_t), 1);
247         if (pnsetp == NULL ||
248             asprintf(&gitcmd, "git --git-dir=%s/.git ls-files", gitroot) == -1)
242         if ((pnsetp == NULL) ||
243             (asprintf(&gitcmd, "git ls-files %s", path) == -1))
249                 goto fail;

251         fp = popen(gitcmd, "r");
252         if (fp == NULL)
246         if ((fp = popen(gitcmd, "r")) == NULL)
253                 goto fail;

255         while (fgets(path, sizeof (path), fp) != NULL) {
256                 newline = strrchr(path, '\n');
257                 if (newline != NULL)
249         while (fgets(fn, sizeof (fn), fp) != NULL) {
250                 if ((newline = strrchr(fn, '\n')) != NULL)
258                         *newline = '\0';

260                 if (pnset_add(pnsetp, path) == 0)
253                 if (pnset_add(pnsetp, fn) == 0)
261                         goto fail;
262         }

264         (void) pclose(fp);
265         free(gitcmd);
266         return (pnsetp);
259         gitmanifest = pnsetp;
260         return;
267 fail:
268         warn("cannot load git manifest at %s", gitroot);
262         warn("cannot load git manifest");
269         if (fp != NULL)
270                 (void) pclose(fp);
265         if (pnsetp != NULL)
266                 free(pnsetp);
267         if (gitcmd != NULL)
271         free(gitcmd);
272         pnset_free(pnsetp);
273         return (NULL);
274 #endif /* ! codereview */
275 }

277 /*
278  * If necessary, change our active manifest to be appropriate for ‘path’.
279  */
280 static void
281 chdir_scmdata(const char *path, const char *meta,
282     pnset_t *(*manifest_func)(const char *path))
269 chdir_hg(const char *path)
283 {
```

```
284         char scmpath[MAXPATHLEN];
271         char hgpath[MAXPATHLEN];
285         char basepath[MAXPATHLEN];
286         char *slash;

288         (void) snprintf(scmpath, MAXPATHLEN, "%s/%s", path, meta);
275         (void) snprintf(hgpath, MAXPATHLEN, "%s/.hg", path);

290         /*
291          * Change our active manifest if any one of the following is true:
292          *
293          *   1. No manifest is loaded.  Find the nearest SCM root to load from.
280          *   1. No manifest is loaded.  Find the nearest hgroot to load from.
294          *
295          *   2. A manifest is loaded, but we've moved into a directory with
296          *      its own metadata directory (e.g., usr/closed).  Load from its
297          *      root.
283          *      its own hgroot (e.g., usr/closed).  Load from its hgroot.
298          *
299          *   3. A manifest is loaded, but no longer applies (e.g., the manifest
300          *      under usr/closed is loaded, but we've moved to usr/src).
301          */
302         if (scmdata.manifest == NULL ||
303             (strcmp(scmpath, scmdata.metapath) != 0 &&
304             access(scmpath, X_OK) == 0) ||
305             strncmp(path, scmdata.root, scmdata.rootlen - 1) != 0) {
306                 pnset_free(scmdata.manifest);
307                 scmdata.manifest = NULL;
288         if (hgdata.manifest == NULL ||
289             strcmp(hgpath, hgdata.hgpath) != 0 && access(hgpath, X_OK) == 0 ||
290             strncmp(path, hgdata.root, hgdata.rootlen - 1) != 0) {
291                 pnset_free(hgdata.manifest);
292                 hgdata.manifest = NULL;

309                 (void) strlcpy(basepath, path, MAXPATHLEN);

311                 /*
312                  * Walk up the directory tree looking for metadata
313                  * subdirectories.
297                  * Walk up the directory tree looking for .hg subdirectories.
314                  */
315                 while (access(scmpath, X_OK) == -1) {
299                 while (access(hgpath, X_OK) == -1) {
316                         slash = strrchr(basepath, '/');
317                         if (slash == NULL) {
318                                 if (!scmdata.rootwarn) {
319                                         warn("no metadata directory "
320                                             "for \"%s\"\n", path);
321                                         scmdata.rootwarn = B_TRUE;
302                                 if (!hgdata.rootwarn) {
303                                         warn("no hg root for \"%s\"\n", path);
304                                         hgdata.rootwarn = B_TRUE;
322                                 }
323                                 return;
324                         }
325                         *slash = '\0';
326                         (void) snprintf(scmpath, MAXPATHLEN, "%s/%s", basepath,
327                             meta);
309                         (void) snprintf(hgpath, MAXPATHLEN, "%s/.hg", basepath);
328                 }

330                 /*
331                  * We found a directory with an SCM metadata directory; record
332                  * it and load its manifest.
313                  * We found a directory with an .hg subdirectory; record it
314                  * and load its manifest.
```



```
333                  */
334                 (void) strlcpy(scmdata.metapath, scmpath, MAXPATHLEN);
335                 (void) strlcpy(scmdata.root, basepath, MAXPATHLEN);
336                 scmdata.manifest = manifest_func(scmdata.root);
316                 (void) strlcpy(hgdata.hgpath, hgpath, MAXPATHLEN);
317                 (void) strlcpy(hgdata.root, basepath, MAXPATHLEN);
318                 hgdata.manifest = load_manifest(hgdata.root);

338                 /*
339                  * The logic in check_scmdata() depends on scmdata.root having
340                  * a single trailing slash, so only add it if it's missing.
321                  * The logic in check_hg() depends on hgdata.root having a
322                  * single trailing slash, so only add it if it's missing.
341                  */
342                 if (scmdata.root[strlen(scmdata.root) - 1] != '/')
343                         (void) strlcat(scmdata.root, "/", MAXPATHLEN);
344                 scmdata.rootlen = strlen(scmdata.root);
324                 if (hgdata.root[strlen(hgdata.root) - 1] != '/')
325                         (void) strlcat(hgdata.root, "/", MAXPATHLEN);
326                 hgdata.rootlen = strlen(hgdata.root);
345         }
346 }

348 static void
349 chdir_git(const char *path)
350 {
351         chdir_scmdata(path, ".git", git_manifest);
352 }

354 #endif /* ! codereview */
355 /*
356  * If necessary, change our active manifest to be appropriate for 'path'.
330  * Check if a file is under Mercurial control by checking against the manifest.
357  */
358 static void
359 chdir_hg(const char *path)
360 {
361         chdir_scmdata(path, ".hg", hg_manifest);
362 }

364 #endif /* ! codereview */
365 /* ARGSUSED */
366 static int
367 check_scmdata(const char *path, const struct FTW *ftwp)
332 check_hg(const char *path, const struct FTW *ftwp)
368 {
369         /*
370          * The manifest paths are relative to the manifest root; skip past it.
371          */
372         path += scmdata.rootlen;
337         path += hgdata.rootlen;

374         return (scmdata.manifest != NULL && pnset_check(scmdata.manifest,
375             path));
339         return (hgdata.manifest != NULL && pnset_check(hgdata.manifest, path));
340 }
341 /* ARGSUSED */
342 static int
343 check_git(const char *path, const struct FTW *ftwp)
344 {
345         path += 2;              /* Skip "./" */
346         return (gitmanifest != NULL && pnset_check(gitmanifest, path));
376 }
```

_____***unchanged_portion_omitted_***