La SHELL di Unix

Schema base

```
int main(void)
{
    int pid, fd, status;
    TOKEN term = T_NL;

    while (true) {
        printf("%s", PROMPT);

        term = command(&pid, false, NULL); //parsing

        if (term != T_AMP && pid > 0) waitpid(pid, &status,0);
    }
}
```

- Questo é uno schema di principio di un progamma di shell: loop infinito con scrittura di prompt e elaborazione della stringa scritta dall'utente (command)
- Se l'ultimo carattere non é & allora aspetto la terminazione. Se é & allora non aspetto e vado avanti: scrivo di nuovo il prompt e il comando viene eseguita in background

Schema base (cont.)

```
static TOKEN command(pid_t *wpid, bool makepipe, int *pipefdp) {
    TOKEN token, term;
    int argc, srcfd, dstfd, pid, pfd[2] = {-1, -1};
    char *argv[MAXARG], srcfile[MAXFNAME] = "";
    char dstfile[MAXFNAME] = "";
    char word[MAXWORD], bool append;
    argc = 0;  srcfd = STDIN_FILENO; dstfd = STDOUT_FILENO;
    while (true) {
        switch (token = gettoken(word, sizeof(word))) {
        case T_WORD:
            strcpy(argv[argc], word); argc++;
            continue;
        case T_GT:
            dstfd = -1; append = token == T_GTGT;
            continue;
        case T_AMP:
        case T_NL:
            argv[argc] = NULL;
            term = token;
            pid = invoke(argc, argv, dstfd, dstfile, append,
                         term == T_AMP, pfd[1]);
            return term;
         case T_EOF:
            exit(EXIT_SUCCESS);
         }
    }
}
```

Schema base (cont.)

```
static void redirect(int dstfd, har *dstfile, bool append,
                     bool bckgrnd) {
    int flags;
    if (dsfd == -1) {
        close(1); flags = O_WRONLY | O_CREAT;
        dstfd = open(dstfile, flags, PERM_FILE); dup(dstfd);
     }
}

static pid_t invoke(int argc, char *argv[], int dstfd,
              const char *dstfile, bool append, bool bckgrnd,
              int closefd) {
    pid_t pid; char *cmdname, *cmdpath;
    switch (pid = fork()) {
    case -1:
        fprintf(stderr, "Can't create new process\n"); return 0;
    case 0:
        redirect(dstfd, dstfile, append, bckgrnd);
        execvp(cmdpath, argv);
        fprintf(stderr, "Can't execute %s\n", cmdpath);
        exit(0);
     }
    /* sono nel processo padre */
    if (bckgrnd) printf("%ld\n", (long)pid); return pid;
}
```

- La redirect effettua la redirezione: crea un file il cui Inode viene messo nella UFDT al posto dello standard out mediante la dup().