Chapter 17

17.1 No solution needed; given in the book.

17.2 No solution needed.

17.3 The following session shows the commands needed to perform the given task. The sample run is given in the book and confirms the proper working of the program.

```
$ cc power.c -o XpowerY -lm
$ XpowerY
[ command output and user input ]
$
```

The following session shows the commands needed to perform the task using the make utility. The sample run is given in the book and confirms the proper working of the program.

```
$ cat makefile
# Sample makefile for the XpowerY program
# Remember: each command line starts with a <TAB>
XpowerY: power.c
cc power.c -o XpowerY -lm
$ make
cc power.c -o XpowerY -lm
$ XpowerY
[ command output ]
$
```

17.4 No solution needed.

17.5 No solution needed.

17.6 No solution needed.

17.7 No solution needed.

17.8 No solution needed.

17.9 Tracking “content” would mean that if in two C program modules with the same exact name, the ASCII text characters on the fifth line where exactly the same. So tracking changes in content would show how the only the characters themselves, without respect to their meaning, on that fifth line differed or remained the same over time after revisions were made.

Tracking “context” would mean that if in two program modules with the same exact name, the fifth line in each had completely different C language syntax on it, different variable names, different structured programming constructs, different argument lists, variable declarations, etc. over time after revisions were made. A meaning and function revision or difference.

17.10 An integrator would take the changes made over time and ensure that they produced a consistent and working build of the software modules being developed independently by team collaborative members. So any conflicts produced by merges of independent collaborators would be arbitrated by the integrator. Content differences indicated by the revision control system and the strategies it deployed, would translate into context differences. The integrator would make sure that the context was the best possible working system for a build of the software.

17.11 Since the Working Directory and .git sub-directory are all contained in a single directory, using the **rm –r** command to delete that single directory works quickly and easily.

17.12 `diff`

17.13 The premise of this section is that we are working with source code that will be compiled. Since object files and executable code are part of the build process, generally only the source code, header files, and make files could possibly be tracked. A separate repository could contain versions of the built system and anything attendant to that, for example tarballs, and other related compiler-mandated modules. A good example of files that would not usually be tracked in Python are the .pyc modules that are generated by the Python Interpreter.

17.14 Yes, because the new text file is an untracked file. So when you checkout any previous commit, you are loading the Working Directory with all of the tracked files at the time of the commit. The untracked ones remain in the Working Directory and are unaffected by the Git database operations, structures, etc..

17.15 Make githubtest/.git the current working directory, and use the command **more config** to examine the refspec assignments. After doing Example 17.3, they should be similar to-

```
[remote “origin”]
        url =https://github.com/bobk48/test.git
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch “master”]
        remote = origin
        merge = refs/heads/master
```

17.16 The command is

`git checkout sha1abbrev`

where sha1abbrev is the abbreviated reference to the commit.

17.17 Make github_clone/test/.git the current working directory, and use the command `more config` to examine the refspec assignments. After doing Example 17.4, they should be similar to-

```
[remote “origin”]
        url =https://github.com/bobk48/test.git
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch “master”]
```

```
        remote = origin
        merge = refs/heads/master
```

17.18 Step 1. Check the remote refs in githubtest-

```
[bob@pcbsd-4976] ~% cd githubtest
[bob@pcbsd-4976] ~/githubtest% cd .git
[bob@pcbsd-4976] ~/githubtest/.git% more config
 [core]
        repositoryformatversion = 0
        filemode = true
        bare = false
        logallrefupdates = true
[remote "origin"]
        url = https://github.com/bobk48/test.git
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
        remote = origin
        merge = refs/heads/master
```

Step 2. Check what files are in the Working Directory of githubtest.

```
[bob@pcbsd-4976] ~/githubtest/.git%
[bob@pcbsd-4976] ~/githubtest/.git% cd ..
[bob@pcbsd-4976] ~/githubtest% ls
README.md   newfile.txt
```

Step 3. Use **git pull** to update githubtest.

```
[bob@pcbsd-4976] ~/githubtest% git pull https://github.com/bobk48/test
master
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 3 (delta 0)
Unpacking objects: 100% (3/3), done.
From https://github.com/bobk48/test
 * branch            master     -> FETCH_HEAD
Updating 4dc2de7..b651617
Fast-forward
 newfile2.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 newfile2.txt

[bob@pcbsd-4976] ~/githubtest%
```

17.19 Here is the command

```
git log --oneline
```

17.20 Make unixthetextbook3/.git the current working directory, and use the command `more config` to examine the refspec assignments. After doing Example 17.5, they should be-

url =https://github.com/bobk48/unixthetextbook3
fetch = +refs/heads/*:refs/remotes/origin

17.21 No solution needed.

17.22 No solution needed.