

GEOS F436/636 Beyond the Mouse

Christine (Chris) Waigl

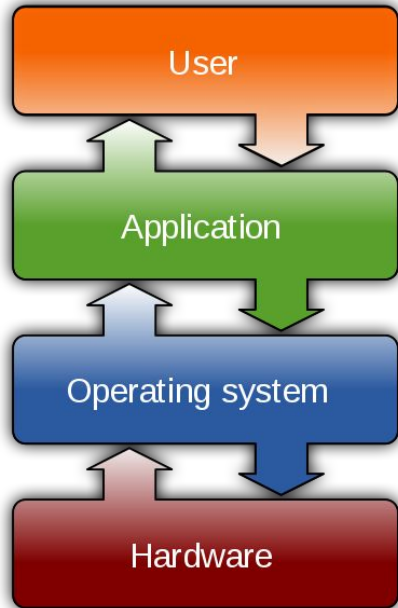
University of Alaska Fairbanks – Fall 2018

Week 9: The Unix command line 1

Topics for week 9

- (Note: Week 8 was two lab sessions, with labs 7 and 8.)
- What is an operating system (OS)? What is Unix/a Unix-like OS and why would we want to use it?
- *ix history and philosophy
- Get your own virtual Linux computer!
- Elementary Unix/Linux command line.

An operating system (OS) is the software that manages a computer's hardware and software resources for the user and application programs.



Thinking questions (take 1-2 min, discuss with your neighbor):

1. Name as many OS as you can.
2. Can you name things that you're not sure whether they are an operating system?

source: Wikipedia

Results from question on previous slide

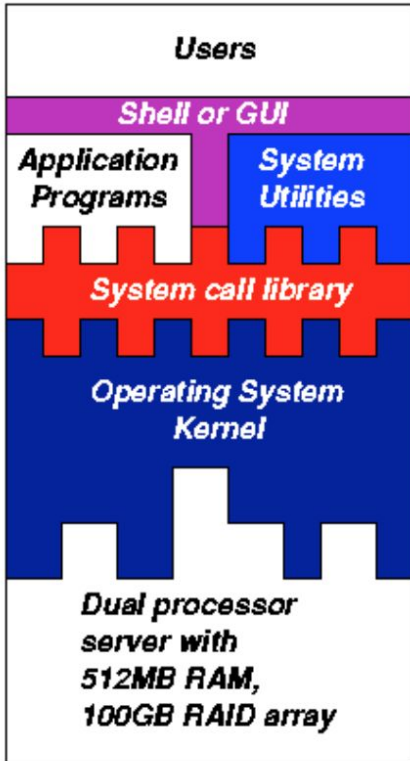
Things that are OSs:

- For desktops/servers : Linux, Unix, Windows (different versions, too -- some just updates of others, some quite different), MacOS, MS-DOS
- For phones and handheld computers: Android (a kind of Linux), Apple iOS, Blackberry
- For network switches and routers: CISCO IOS for example

Things that aren't OSs:

- "Microsoft" -- is a company that makes OSs and other software
- "Oracle" -- is also a company, but more importantly a database management system. They are in some ways similar to OSs (infrastructure that the user doesn't directly use), but don't manage computing hardware/resources

Frequently found functions of an OS



All the parts in color on the left (source: drexel.edu) can be considered part of the OS:

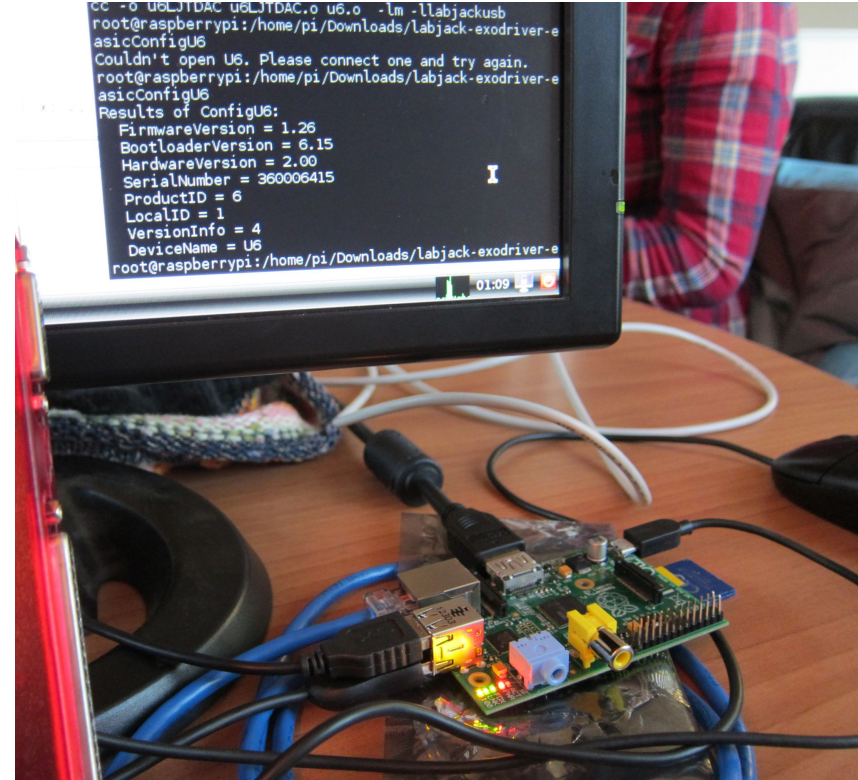
- The kernel is the most central part. It is loaded first at start-up and manages memory and processes.
- The kernel loads device drivers to interact with the hardware
- A library of system and a set of system utility programs calls may be available to applications to exchange data and commands with the system.
- A shell (command line) or a GUI offers the user an interface to interact with the system.

There are many different specialized OS

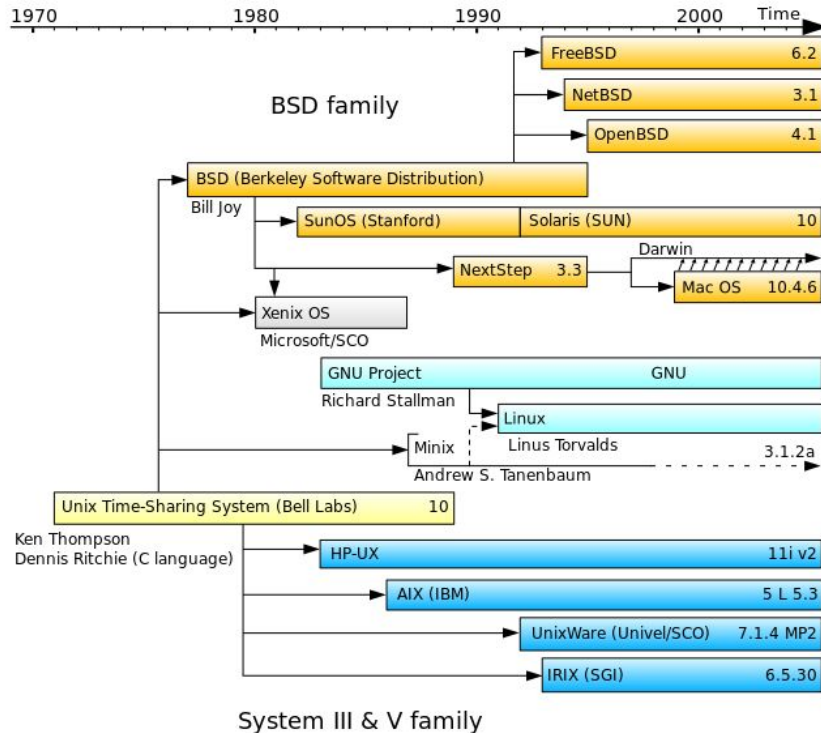
- For real-time systems, embedded systems, workstations, mobile systems...
- Early computers were specialized calculating machines. As hardware capabilities increased, computers were designed with management software to "supervise" the execution of programs and the access to resources.
- But even with mainframes (into the 1970s and beyond), computing meant basically a) load a program from magnetic tapes or plates and b) execute the program. ⇒ Batch mode
- In parallel, time-sharing OS allowed several processes, and later users, to be allocated small slivers of resources (nearly) simultaneously.
- Unix, which started in the 1970s at Bell Labs as a project by Brian Kernighan and Rob Pike became a very successful time-sharing multi-user OS.

Why would you like to use Unix/*ix ?

- Use (nearly) any large modeling system (WRF, climate model, tsunami model, ice sheet modeling...)
- High-performance computing for science (NCAR, Chinook supercomputer)
- GNU/Linux offers a free, collaboratively developed computing environment that you can use as your primary machine!
- Embedded systems with Raspberry Pi (free small computers running GNU/Linux) for research, sensors, robotics, autonomous cameras....



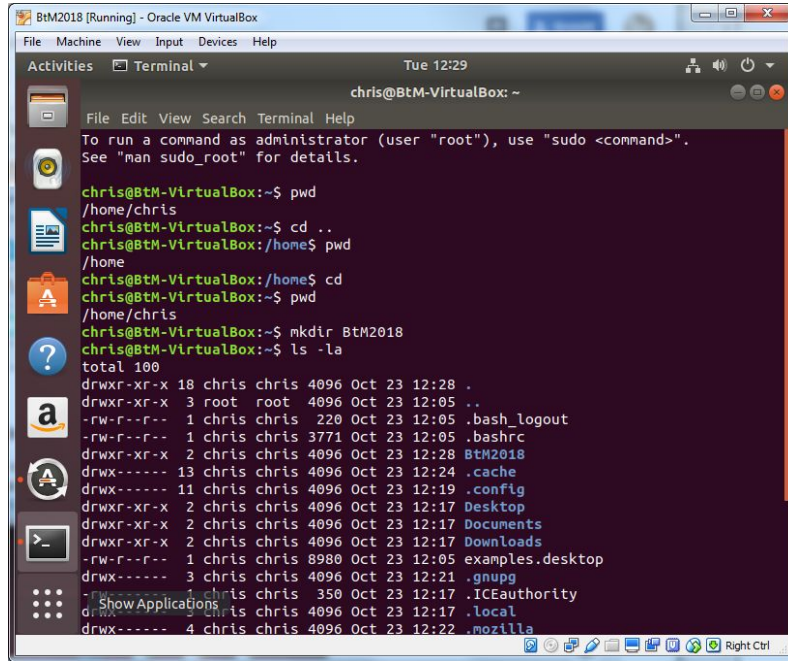
A very very simplified history of Unix



The researchers at Bell Labs defined "what Unix is", and many others implemented versions of it.

- orange: open-source
- dark blue: proprietary/commercial
- turquoise: the GNU/Linux system (GNU system utilities + Linux kernel written by Linus Torvalds and collaborators)

We will now create and install our very own Linux system in a virtual machine on your Windows computer. (See stand-alone instructions.)



The screenshot shows a terminal window titled "BtM2018 [Running] - Oracle VM VirtualBox". The terminal displays the following commands and output:

```
chris@BtM-VirtualBox: ~
File Edit View Search Terminal Help
Tue 12:29
chris@BtM-VirtualBox: ~
File Edit View Search Terminal Help
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
chris@BtM-VirtualBox:~$ pwd
/home/chris
chris@BtM-VirtualBox:~$ cd ..
chris@BtM-VirtualBox:/home$ pwd
/home
chris@BtM-VirtualBox:/home$ cd
chris@BtM-VirtualBox:~$ pwd
/home/chris
chris@BtM-VirtualBox:~$ mkdir BtM2018
chris@BtM-VirtualBox:~$ ls -la
total 100
drwxr-xr-x 18 chris chris 4096 Oct 23 12:28 .
drwxr-xr-x  3 root  root 4096 Oct 23 12:05 ..
-rw-r--r--  1 chris chris  220 Oct 23 12:05 .bash_logout
-rw-r--r--  1 chris chris 3771 Oct 23 12:05 .bashrc
drwxr-xr-x  2 chris chris 4096 Oct 23 12:28 BtM2018
drwx----- 13 chris chris 4096 Oct 23 12:24 .cache
drwx----- 11 chris chris 4096 Oct 23 12:19 .config
drwxr-xr-x  2 chris chris 4096 Oct 23 12:17 Desktop
drwxr-xr-x  2 chris chris 4096 Oct 23 12:17 Documents
drwxr-xr-x  2 chris chris 4096 Oct 23 12:17 Downloads
-rw-r--r--  1 chris chris 8980 Oct 23 12:05 examples.desktop
drwx-----  3 chris chris 4096 Oct 23 12:21 .gnupg
-rw-r--r--  1 chris chris  350 Oct 23 12:17 .ICEauthority
drwx-----  4 chris chris 4096 Oct 23 12:17 .local
drwx-----  4 chris chris 4096 Oct 23 12:22 .mozilla
```

The end result looks like the image on the left.

The user interacts with the computer via a GUI (like in Windows) or via a very powerful command line interpreter called a **shell**.

Command prompt:

```
chris@BtM-Virtualbox:~$
```

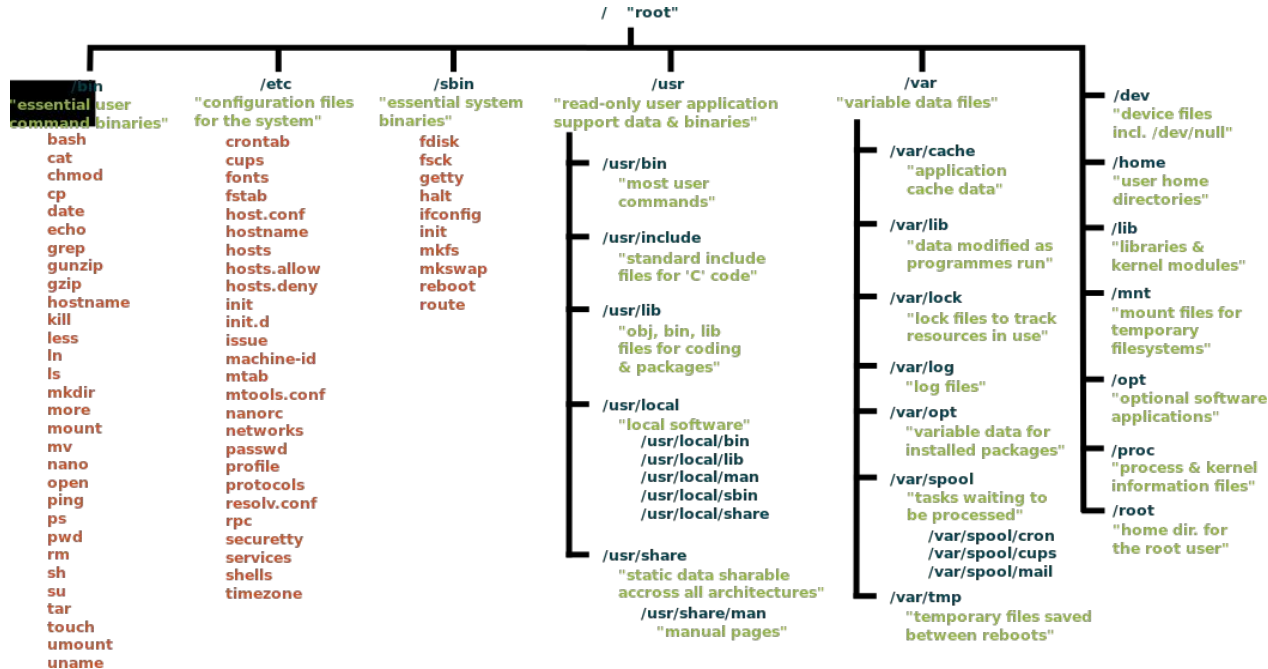
user@computer:path\$ ← Command prompt pattern. The user can decide what this should look like. (We will see how next week.)

The famous "Unix philosophy": Provide tools that do one thing and do it well.

You can read more about it on Wikipedia https://en.wikipedia.org/wiki/Unix_philosophy or in the (still excellent) 1984 book *The Unix Programming Environment* by Kernighan and Pike:

*Even though the UNIX system introduces a number of innovative programs and techniques, no single program or idea makes it work well. Instead, what makes it effective is the approach to programming, a philosophy of using the computer. Although that philosophy can't be written down in a single sentence, at its heart is the idea that the power of a system comes more from the relationships among programs than from the programs themselves. **Many UNIX programs do quite trivial things in isolation, but, combined with other programs, become general and useful tools.***

Unix-like systems organize files in a hierarchical tree



Each directory has a path. Home directory (where you log into):

`/Users/chris` OS X

`/home/chris` Linux

The home directory is called `$HOME` or `~`.

The Unix shell

- The command line interpreter we interact with is called a *shell*
- There have been many shells developed for Unix systems over the years. The original shell called sh ("Bourne shell") is still available on most systems.
- We will use the default shell that comes with Ubuntu GNU/Linux: bash ("Bourne Again SHell" ← Attention, JOKE!!)
- A shell is really just another program, like all Unix tools! We can easily change it. Try typing `tcsh` at the command prompt. (Get out of `tcsh` with CTRL-D.)
- The shell is like a REPL (read-execute-print-loop) in MATLAB. It prints to **standard output** and reads from **standard input**

Unix commands 1: Moving around the tree

- \$ `pwd` ← "print working directory" (ie the directory where you currently are: "Where am I???")
- \$ `cd ..` ← "change directory" (.. = one up)
- \$ `cd /` ← go all the way up to the root directory (absolute path)
- \$ `cd ~` (or simply: `cd`) ← go to your home directory
- \$ `cd $HOME` ← same thing
- \$ `mkdir newdir` ← make a new directory called newdir inside your current dir.
- \$ `cd newdir` ← go into your new directory (relative path: one down)
- \$ `cd ./newdir` ← same thing: . is the current directory
- \$ `cd ../otherdir` ← go one up, and then down into otherdir (if exists)
- \$ `cd /usr/bin` ← go to an absolute path /usr/bin

Unix commands 2: Commands can have options and arguments (eg. list directories (ls), print (echo))

Anatomy of a command:

`$ ls -la` ← command-line options. short for -l (long format)
-a (show all files, including hidden files)
long options (more than 1 letter) start with --
↑
↑
command
(end of) command prompt
(you don't have to type this)

`$ echo hello world` ← argument

`$ man command` ← get "manual page" of command

You create a hidden file or directory by starting the file name with a dot. For example: `.bashrc` in your home directory.

Unix interlude 1: Users

- You created a username and a password for yourself during installation
- You can have as many users as you like! See the `useradd` command.
- Users are members of *groups*. This is to manage sets of permissions that apply to a whole range of users. Groups are useful for example if you want to give all members of your research team access to some data or resources. We won't talk more about them than necessary. You can type `groups` to find out which groups your user is part of.
- `root` is a special user, also called the *superuser*, with total control. For tasks that require root privileges (for example, install software), you can either "become" the root (but this is disabled by default on Ubuntu Linux), or use the `sudo` command ("do as supersuer", pronounced "soodoo".)

Unix interlude 2: File and directory permissions

When you use the long form of ls to list all files and directories in a directory, you'll see something like this:

```
chris@BtM-VirtualBox:~$ ls -la
total 100
drwxr-xr-x 18 chris chris 4096 Oct 23 12:28 .
drwxr-xr-x  3 root  root 4096 Oct 23 12:05 ..
-rw-r--r--  1 chris chris  220 Oct 23 12:05 .bash_logout
-rw-r--r--  1 chris chris 3771 Oct 23 12:05 .bashrc
drwxr-xr-x  2 chris chris 4096 Oct 23 12:28 BtM2018
drwx----- 13 chris chris 4096 Oct 23 12:24 .cache
drwx----- 11 chris chris 4096 Oct 23 12:19 .config
drwxr-xr-x  2 chris chris 4096 Oct 23 12:17 Desktop
drwxr-xr-x  2 chris chris 4096 Oct 23 12:17 Documents
```

d: directory

rwxrwxrwx: read/write/execute permissions for user (owner), group, and everyone else

chris chris: owner user and group

+ id, file size, time stamp, file/directory name.

(Here, directories are displayed in blue) (Do not worry at this stage about the "group" thing. The important bit is that permissions to read, write and execute a file can be set individually.

"Execute" means "run" if it is a program, "access" if it is a directory.

Examples for permissions

File Attributes	Meaning
-rwx-----	A regular file that is readable, writable, and executable by the file's owner. No one else has any access.
-rw-----	A regular file that is readable and writable by the file's owner. No one else has any access.
-rw-r--r--	A regular file that is readable and writable by the file's owner. Members of the file's owner group may read the file. The file is world-readable.
-rwxr-xr-x	A regular file that is readable, writable, and executable by the file's owner. The file may be read and executed by everybody else.
-rw-rw----	A regular file that is readable and writable by the file's owner and members of the file's group owner only.
lrwxrwxrwx	A symbolic link. All symbolic links have "dummy" permissions. The real permissions are kept with the actual file pointed to by the symbolic link.
drwxrwx---	A directory. The owner and the members of the owner group may enter the directory and, create, rename and remove files within the directory.
drwxr-x---	A directory. The owner may enter the directory and create, rename and delete files within the directory. Members of the owner group may enter the directory but cannot create, delete or rename files.

Symbol	Meaning
u	Short for "user" but means the file or directory owner.
g	Group owner.
o	Short for "others," but means world.
a	Short for "all." The combination of "u", "g", and "o".

Notation	Meaning
u+x	Add execute permission for the owner.
u-x	Remove execute permission from the owner.
+x	Add execute permission for the owner, group, and world. Equivalent to a+x.
o-rw	Remove the read and write permission from anyone besides the owner and group owner.
go=rw	Set the group owner and anyone besides the owner to have read and write permission. If either the group owner or world previously had execute permissions, they are removed.
u+x, go=r-x	Add execute permission for the owner and set the permissions for the group and others to read and execute. Multiple specifications may be separated by commas.

Changing permissions:
chmod a+rwx [file_name]

Octal notation for each
of the three symbols:
chmod 777 [file_name]

Octal	Binary	File Mode
0	000	---
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx

Unix commands 3: Manipulating files

\$ cp file1 file2 ← "copy" (make a duplicate of file1 named file2)

\$ mv file1 file2 ← "move" (= rename) (cp and mv work across directories!)

\$ mv file1 dir1/ ← move file1 into directory dir1

\$ rm file1 ← "remove" (delete forever, without hesitation!)

\$ rm -r dir1 ← delete dir1 and everything in it (recursively)

\$ touch file1 ← create empty file1. if exists: reset timestamp

\$ cat file1 ← print the contents of file1 to standard output

\$ more file1 (or: less file1) ← print the contents of file1 page by page

\$ wc file1 ← print number of lines, words, bytes in file1

\$ wc -l file1 (or: wc --lines file1) ← print number of lines

\$ chmod o+rw [file] ← "change mode" give "others" read/write permission

\$ tail (or: head) file1 ← show last (or: first) lines of file1

Unix commands 4: Input/output, redirection & pipes

`$ echo hello world` ← print the argument ("hello world") to standard out(put)

`$ cat file1 > file2` ← redirect contents of file1 into file2 (overwrites)

`$ cat file1 >> file2` ← append contents of file2 at the end of file2

`$ echo This is the last line >> file1` ← appends last line to end

`$ echo There are five words here | wc -w` ← pipes the output of the echo command ("There are five words here" into the command `wc -w`, which counts the words. The result is the number of words: 5.)

`$ cat file1 | wc -l` ← counts the lines in file1

`$ ls -l > listing.txt` ← create a file listing.txt that contains the directory listing

`$ ls -l | wc -l` ← counts the number of files & directories (lines in a long

directory listing)

Unix commands 5: Information about your system

Unix has many commands that tell you about your system. Most of them have many options due to the fact that Unix is much used in complex large systems, so it is designed to be easy to use for system administrators (sysadmins).

- \$ `uname -a` ← OS information
- \$ `hostname` ← your computer's name
- \$ `uptime` ← how long your computer has been running ("up")
- \$ `whoami` ← your user. Also try `w` and `who`
- \$ `top` ← show all running processes and their resource usage
- \$ `ps` ← information about processes in particular command and process ID (PID)
- \$ `kill pid` ← kill (stop) the process with the id `pid` `kill -9` kills harder
- \$ `killall command` ← kill all processes that start with the command "command"

Unix interlude 3: Keyboard shortcuts, wildcards etc.

If you launched a process and need to get back to your shell prompt, and in other situations, there are a number of keyboard shortcuts that can help.

`CTRL-C` (hold CTRL and C keys simultaneously) abort command execution

`CTRL-D` send "end-of-file" message: ends shell, ends other REPLs

`q` quit, gets you out of more, less and man pages

`$command &` sends the command execution to the background,

`CTRL-Z` followed by `bg`: halt command, then send it to the background

`fg` return the last command sent to the bg back to the foreground

`*` matches an arbitrary text. Eg: `*.txt` = all files ending in `.txt`

`?` matches one character. Eg: `file?.txt` matches `file1.txt`, `file2.txt`, `filea.txt`, `fileb.txt` ...