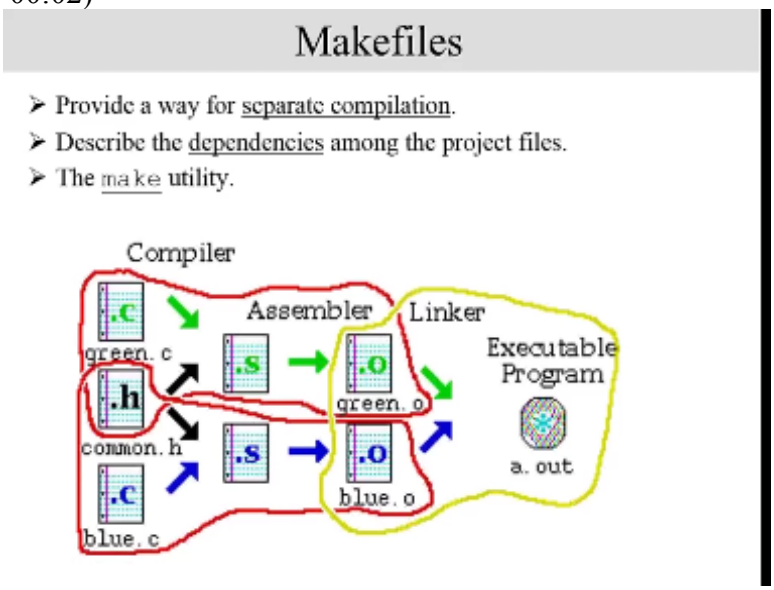
(Refer Slide Time: 00:02)



Hi everyone again welcome to this session of the LPS today probably will be the last lecture we have covered so many things over the last several weeks started with looking at the Linux then went into programming we started with Perl we covered TCL then we also covered Python and today we are going to talk about new files are one of the e-utilities or the running net program automatically, the reason why we move make else is because we want to capture the dependencies.

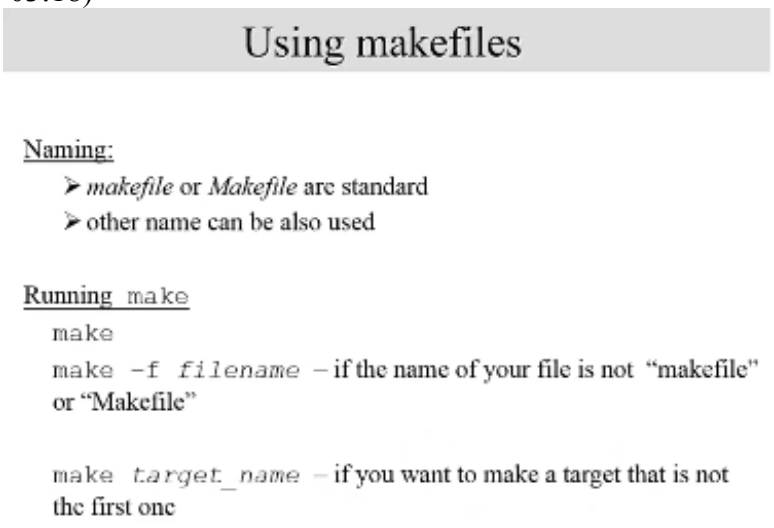
And essentially like new the dependency to do certain things as well as not do certain things , so in a serial processing of files basically if you are doing something to one side and then you are going to doing something else and if you if there is an error occurring at one point maybe you need to back out to some place before you can continue at the same time if you find some error and then you have done some already like you processing steps.

You do not want to go back all the way the beginning and then do the first thing all over again you want to capture from the middle wherever it is if it is successful from that point onwards you one more forward and for these kind of things and make files lists are used there you can capture the dependencies in a serial fashion basically and then you can use that to your advantage.

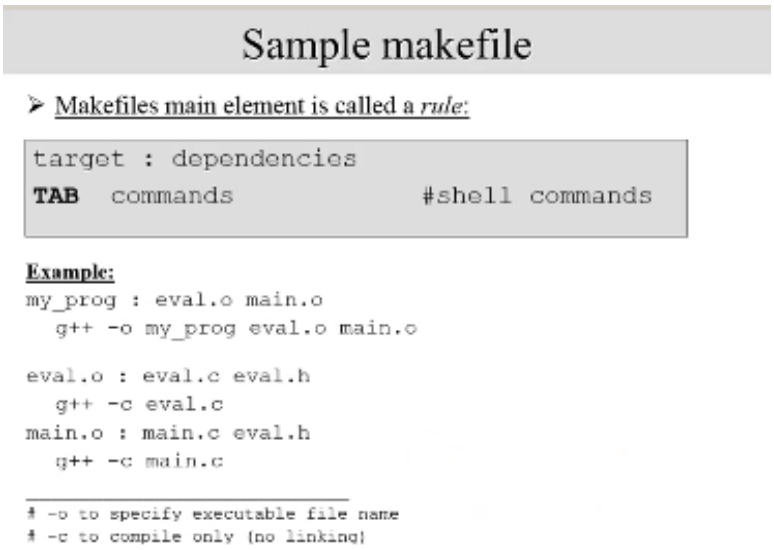
In terms of what needs to be done and what does not need to be so a make file is forwarding away for a separate compilation describe dependencies, among the project plans and then that is the make you can use the make you ability to do the comparison ,so in a typical process for compilation.

You will have like meaning number of programs click .C and they will probably share a common header file from the H common. H and then now so this is basically in a typical C environment

where you have multiple stuff and then basically like they are assembled ethnicity and then from the assembler you generate the part over the executable. which essentially link it endowing can link it and then you can convert this program so now if you want to change the C but you do not I mean the green dot C but not the blue dot C.
You do not want to compile the blue .T again so you only want to compile the theme the green dot C so a make file once you specify how they are dependent it looks at the timestamp and looked at what changed and it is nothing as change then it would not touch the particular file.
(Refer Slide Time: 03:18)



So let us look at that so using the make file basically lights also in a file or make five uppercase M definitely look at that those are kind of ones even you can do other file powerful and to run the make it basically simply the make is the commanding and then you can tell like make – f file name if the name of the file is not going to be fun otherwise it will just run with me fine and then you can also say like make a target mean which is a more kind of target that you want to meet.
(Refer Slide Time: 03:59)



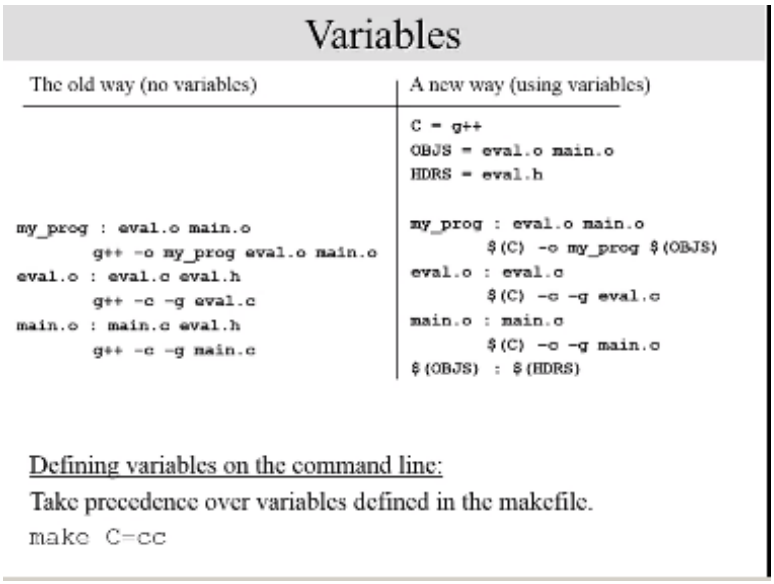
If the target is not included in the make file itself, so a simple make a sample made file this is essentially like a mean or the main element in the sample the rule of the main file import folder the rule is simply target : dependencies and then you can also like use tab or any kind of commands to get the particular type, so it is an example so the target is what is called my in the corporal and then the dependencies are a valve .O and main .O so the top defined these two.

Now once these dependencies are satisfied which command so here we run the D ++ which is a compiler and then - o it is the output will be remind prog and then the inputs are eval dot o and now we go hierarchically and describe each one of the dependencies so for example, eval .O depends on eval .C.

And evaluate .H this is essentially you can think of this as a green. P and the image the previous one in the eval. H is the error and then how do we get to the eval .O basically we do a C++ compiler and eval .C so this will generate the eval .O now let us look at the mean .O the main auto again its components or the dependencies are main . C and eval. H is all alright you notice that they are thing from both sides.

And now the command to generate the main .O is G ++ and then we say - compile -P and then we do main. C, so we can also specify the comments using hashes that we already know it gives so the syntax is very similar to any other feedback regular T programming feedbacks ,you can say so basically like the - O if this executable file name –P compile no link so the - O does the compiler thinking of V – basically.

(Refer Slide Time: 06:50)

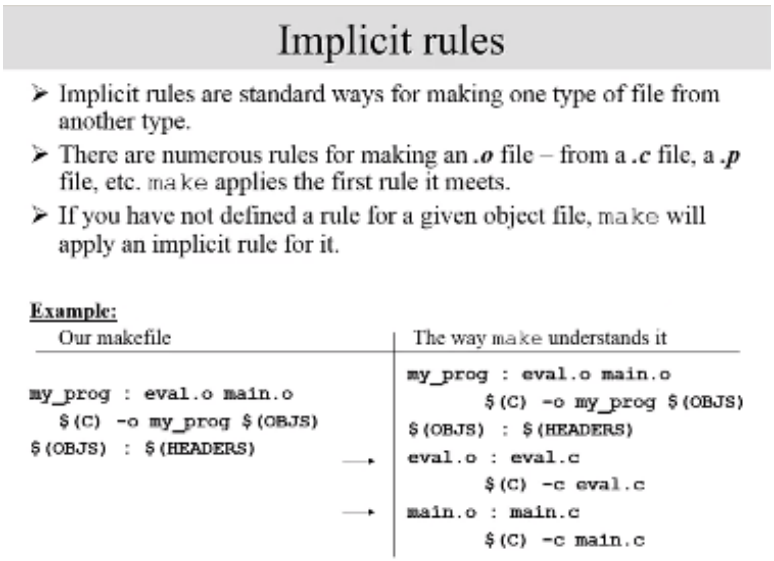


So you can also make a file with programming constructs such as variable user variables so here the old way we figure on the left end time that you see it until it there we specify the top-level target and then each dependencies and then we go have a higher and define differences visit exact same activity for the my prog but if you want to use the variable then we can define the variable constant at the head of so for example, C is defined as V++OBJ of the object is eval .O and main .O and then the headers are eval . H.

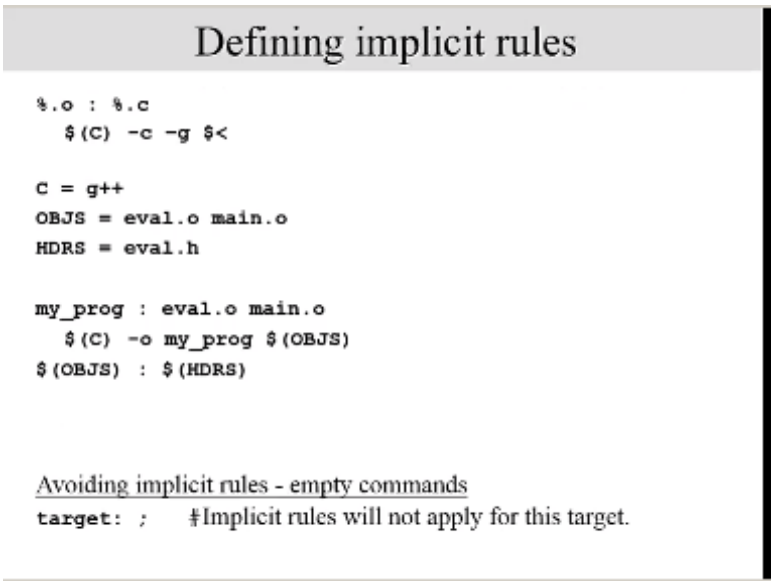
so now we simply contain that my frog is eval .O main .O that is the dependencies but the command is essentially like dollar P this is D ++ and then the arguments are for the input is basically like dollar of PS which is these two inputs and then we also now go and do the same thing again for eval.O all this is the subsequent targets and then the main .O and then here we define the key like the one more additional one which is the dollar obj s it is target basically or if dependent P is located.

Which is the eval. H and essentially like I mean so only making on develop eval. H present then it starts working on working there so this is convenient mainly because now if you want to replace the D ++ compiler it a CT compare go take a compiler then all you want to do is this change that thing and in fact you do not even have to change it inside of a file, but just define it like make t = cc and then automatically like everything is fitted with CP and though they take presents over the variables defined inside of client dynamic so finally simple enough.

(Refer Slide Time: 09:10)



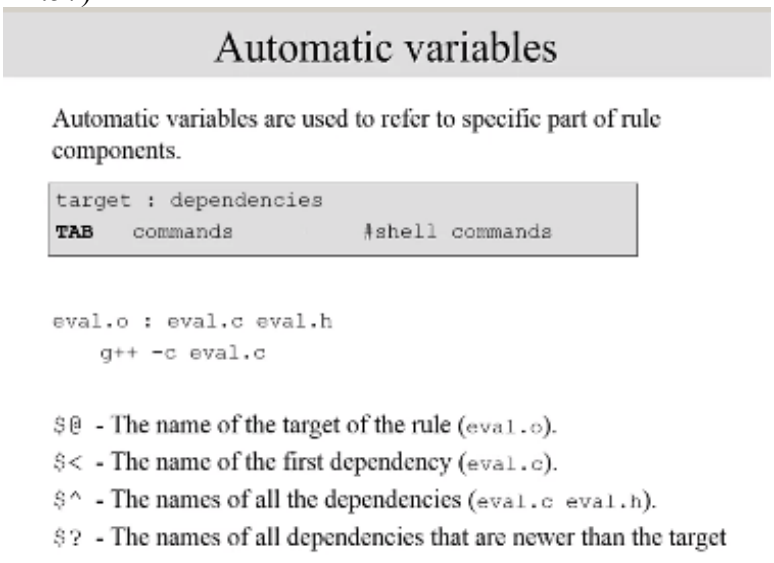
And now there are some implicit rule essentially the implicit rules are standard for making one type of file from another type there are numerous rules for making .O file from a. B file from a dot T file a etc.. And make usually applied the very first rule that means if you have not defined the rule for a given object file make will apply an implicit over for example our make file please specify basically like eval. O main .O and then we say like them taller see and so Prog the objects and then ball of this dollar fit is so here we omitted the eval .O and main. O target.
So it applied the implication and then basically creates its equivalent of creating this rule where it is eval .O depends on eval .C and then this is comparative eval .C and main. O compare eval. C.
So this is the way that the make understand this small Meet which indicate it is okay but make sure that if some something that if you are not explicit about it make file will assume and then start reporting this way.
(Refer Slide Time: 10:39)



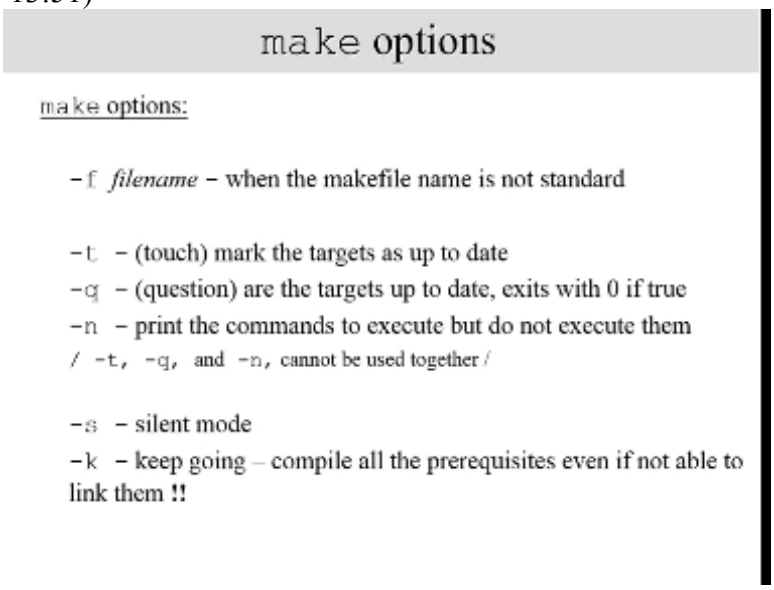
Now let us look at another way basically visible using the percentage, percentage .O is percentage. C and then this is a very way of representing a rule and essentially like. I mean again here this is it that replaces all these things with the profit stuff and then course if it works its way to do the me ,so here you can see that once you specify that and then now basically they give the same thing but at the same time if you do an empty command then it would not apply the implicit rule.

So it is as good as specifying a rule and then specifying and empty command so just take care if you do not want any rule to be applied then you can specify that an empty commands and that really the implicit rules will not get applied , so that is one way to empty command.

(Refer Slide Time: 11:57)



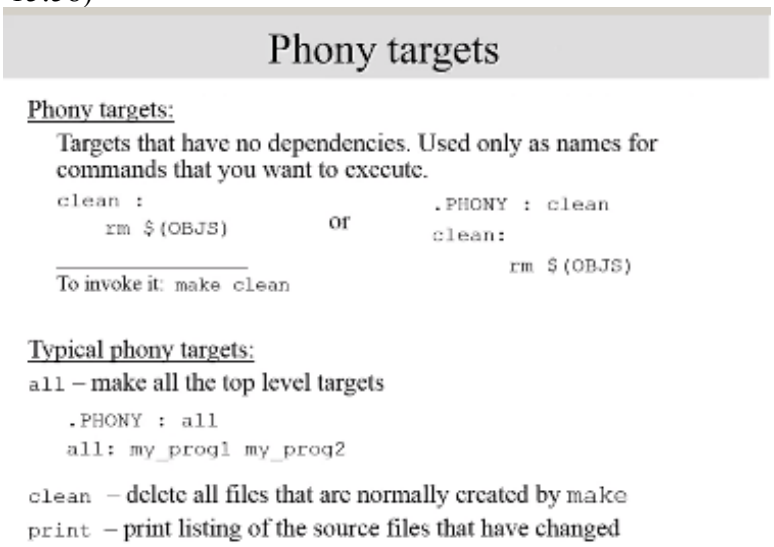
So now there are some variable you calling in the previous one to use the dollar less than so those are what is called the automatic variables the automatic variables are used to refer to the specific part of the rule components , so we know that basically like target dependencies at the general structure of the room and then followed by tab and then the commands those are them would not be a shell command.
So here you can pay basically like eval .O eval .C and eval .H and then V++ is the command and then basically eval. C is the angular so the dollar at will determine the name of the target of the rule in this case it is eval. O and then the dollar left an actually specifies being first dependency this one it is eval. C so now you should go back and E with B when we specify like dollar less than having the person dependency which is the percentage .C now dollar target is name of all the dependencies.
That is in this eval. C and eval. H and then the dollar question mark names of all the dependencies that are where than the target so in this case actually like I mean the nothing in this one so basically like there is nothing that it is defined for the dollar question.
(Refer Slide Time: 13:51)



Now let us look at some of the make options -F specifies the finding which is if the make file fill the name is not just a make file so if you certify this the make file then give or need to specify the null back there otherwise you need to specify -N and then - T is essentially using the command called touch to mark the targets as up to date and then the - Q is also known expected basically it looks for whether the target are up today and if it is true then it exists to the 0 and -N is just the print the command to execute.

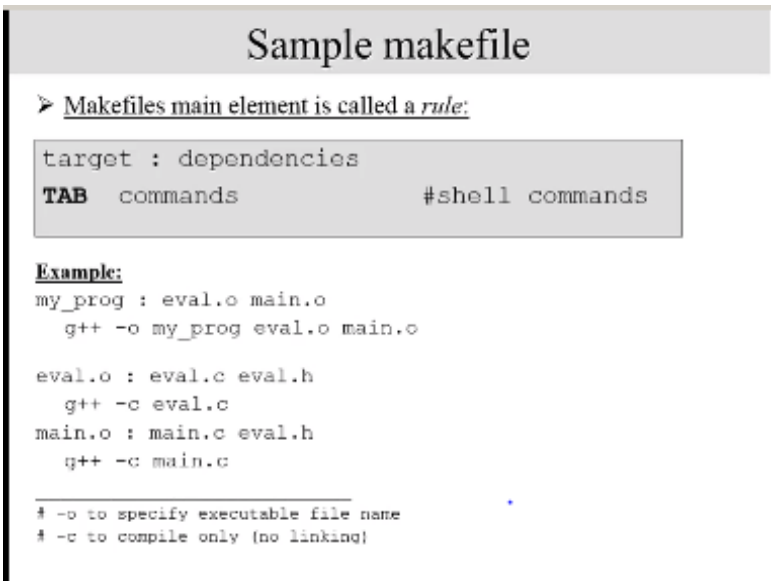
But do not really do so if you do like a arm - N that make - N of your main file which is maybe like this it will only like print all the various commands will give it - t is like G+ class and so prog then it also kept a deeper / - eval.C and then the D++ - C main .C and it is link so one teenage like I mean so this is one way to actually just spit out whether your dependencies are correct this do a -N and then one thing to notice this with that –P-Q-N cannot be used together so the only one of them should and then - S is silent mode that is it runs basically without echoing what is secure on and then - K is keep going waiting meaning comparing all the prerequisites even is it is not able to link them .

(Refer Slide Time: 15:58)



Now in order to make a flow work you need to also specify some phony targets the phony targets are targets that have no dependable they are used only as name for commands that you want it skip so there are no dependencies with so you are because basically kind of pillar for example, there is one target called clean which is set to remove all the files ,so you can also specify a long time with it is got phony is steamed and then pain is you can specify that clearly as the pain is any target and then to invoke this you can get pay make in and then it will turn that so one thing to notice like.

(Refer Slide Time: 17:01)

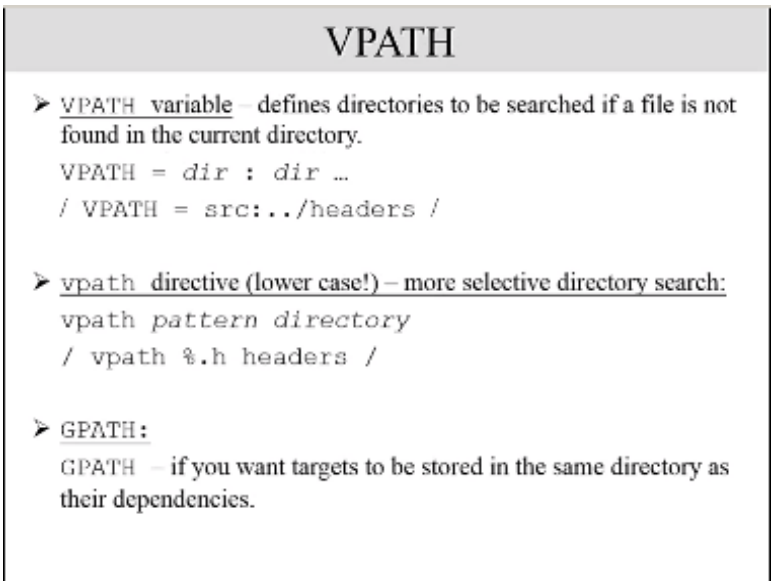


I mean I mentioned here once you have this is your make file you can only run subsections of movement for example you can stay like make give eval .O and then it will only run this command the eval .C eval. H and then D ++ actually only the one starting from the eval .O and then goes down for example, it know the targets basically the dependencies are eval.C and eval .H and then it will run the D + + - t eval .C eval .H if this looks for the File with a disc event then it comes this command but then it goes to the next main .O that mark is not a dependency so oil dependency for eval .O .

So it will stop there same thing you can run just main.O and then it will run this third thing and then it will stop only if you specify the target as my prog then it will run everything that - only if there is any change in the eval .O and main .O then it will start running it will evaluate the dependency the dependencies are okay, basic phony targets.

So here similarly like I mean if you want this to move the target team with the same 18 and then it picked a removes all the time and then the typical Phony targets basically without like to make all the top-level targets so we call it like phony all and then always all the targets basically are there so if you do a make all it runs with a everything been whether everything okay, and then the other Phony target is the screen which deletes all the files, which are normally created by make and then print with this printing between the listing of the source files that have changed.

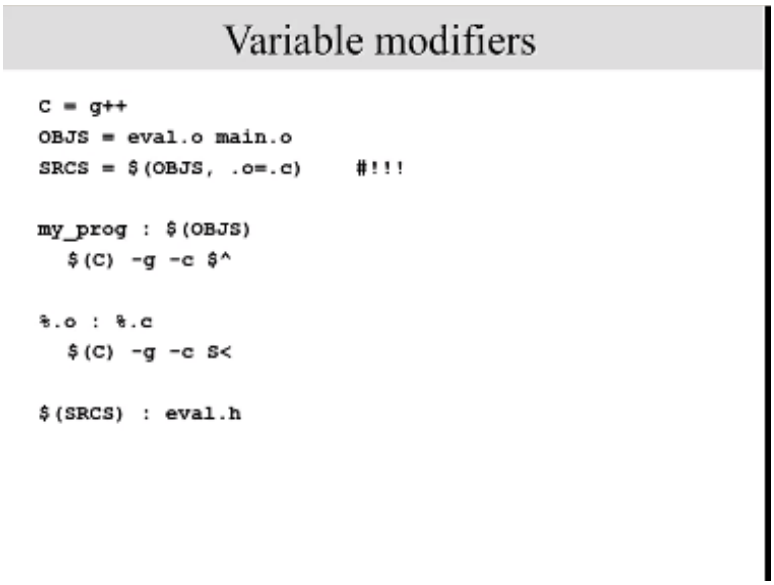
(Refer Slide Time: 19:09)



But now another thing is basically VPATH variables because a variable that defines directory is to be searched if a file is not found in the current directory and here you can specify a number of benefits separated by columns ,so for example here VPATH equal to the colander : therefore all basically the first one it is taken as a source and the second one is actually the fact to that particular directory, so you can say back your colander this is one of them and then you can tell like Colin the to another one.

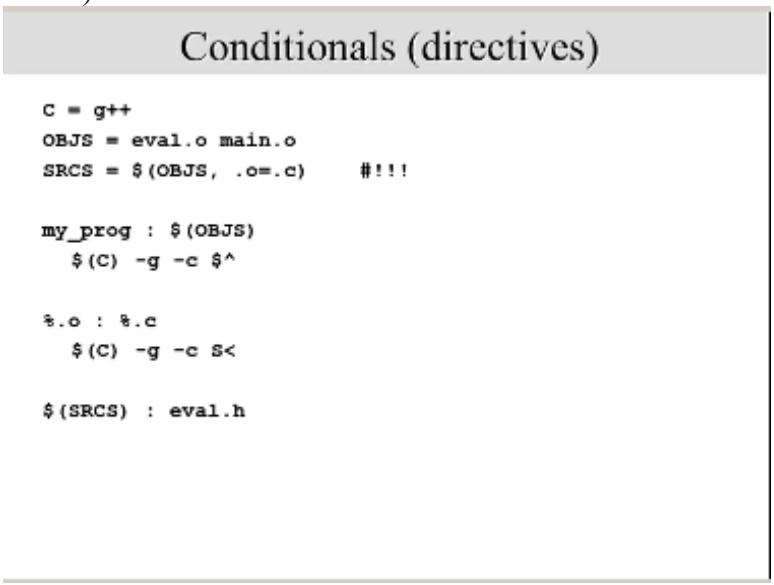
So similarly separated one separated many as not and then if you want to specify like a directive like the lower case VPATH that is the Selective directory search then you specify a pattern and then what the directory is for example, with your VPATH person based on H and then that is in headers now similar to be part there is always GPATH if you want the targets to be thrilled with pain directly as their dependency. You specify the default.

(Refer Slide Time: 20:46)



Now there is also some variable modifying it somewhere eventually the objects essentially this is the eval .O main.O and then we do a compile and then you know that the dollar target represents all the targets so that evil .O main .O now the sources actually is essentially the payment of this only thing you will be persecute the .O with. C that is what the syntax means, so when we specify that here we can just pay forces if eval .H basically where those are the source the targets second C eval.H.

(Refer Slide Time: 22:17)



And we can also use conditionals to change the way that the make file target get executed , so we can specify specific things basically and then the possible conditionals are if EQ if any Q is def idndef and all of them should be closed within endif and then the conflict one can be either new

to that Elif or else so here is one example, so here we can specify basically like before it is a GCC or something else some other compaction and for GCC.
We can refer the library that - LGNA but the normal libraries are dependable so now if you say basically like the if dollar PP GCC that is dollar series DC then like this dollar life GCC tell its life is equal to dollar normal it and end it so noting that actually there are no tabs at the beginning these are basically this assignment.
So it automatically like I mean now when you specify the particular target and the dependency and then the actual command based on what get executed each automatically we created this one, so that is pretty much what I have on my files if you have any other cards to share please do so with your TA and thank a lot okay.